On design and analysis of symmetric key ciphers

Nguyen, Phuong Ha

2013

Nguyen, P. H. (2013). On design and analysis of symmetric key ciphers. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/51183>

<https://doi.org/10.32657/10356/51183>

ON DESIGN AND ANALYSIS OF SYMMETRIC KEY CIPHERS

NGUYEN PHUONG HA

SCHOOL OF PHYSICAL & MATHEMATICAL SCIENCES

2013

ON DESIGN AND ANALYSIS OF SYMMETRIC KEY CIPHERS

NGUYEN PHUONG HA

School of Physical & Mathematical Sciences

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2013

ACKNOWLEDGEMENTS

This thesis could not have been completed without the help of many people to whom I am indebted.

First and foremost I would like to express my heartfelt gratitude to my supervisor Professor Wang Huaxiong. His understanding, encouragement and personal guidance have helped me to complete this thesis.

Special thanks go to Professor Ling San, Professor Wu Hongjun, Dr Matthew John Barton Robshaw and Professor Axel Poschmann for their supervision, inspiration, and collaboration across my entire PhD program.

I wish to express my warm and sincere thanks to my coauthors WeiLei, Chester Reibero, Sebastian Thomas Kutzner and Professor Debdeep Mukhopadhyay.

I am deeply grateful to Le Tuan Quy Ngoc, Xinli, Dau Son Hoang, and Dr Ivica Nikolic who have helped me editing my thesis.

I thank my fellow labmates Guo Jian, Weilei, Ding Yang, Chen Jie, Zhang Liangfeng and Zhang Yun who have shared a much treasured memory here in SPMS-MAS-04-01, NTU with me.

I would also like to thank my friends D.T.T, H.T, T.L.K, T.H who have supported me during the PhD program.

I warmly thank Nguyen Anh Chi, Do Van Thuy Duyen who have given me continuous support.

And, above all, my family - my father Nguyen Phuong Son, my mother Nguyen

Mai Hong and my younger brother Nguyen Phuong Mai; without their unconditional love and understanding, I won't be where I am today and my aspirations could not possibly been achieved.

PREFACE

All chapters in this thesis have been published or submitted in different papers. The following is the list of these publications:

LIST OF PUBLICATIONS

1. Phuong Ha Nguyen, Lei Wei, Huaxiong Wang, San Ling: On Multidimensional Linear Cryptanalysis. ACISP 2010: 37-52
2. Phuong Ha Nguyen, Hongjun Wu, Huaxiong Wang: Improving the Algorithm 2 in Multidimensional Linear Cryptanalysis. ACISP 2011: 61-74
3. Phuong Ha Nguyen, Matthew J. B. Robshaw, Huaxiong Wang: On Related-Key Attacks and KASUMI: The Case of A5/3. INDOCRYPT 2011: 146-159
4. Phuong Ha Nguyen and Chester Rebeiro and Debdeep Mukhopadhyay and Huaxiong Wang: Improved Differential Cache-Trace Attacks on SMS4. IN-SCRIPT 2012.
5. Sebastian Thomas Kutzner, Phuong Ha Nguyen, Axel Poschmann and Huaxiong Wang: On 3-share Threshold Implementations for 4-bit S-boxes. COSADE 2013.
6. Sebastian Thomas Kutzner, Phuong Ha Nguyen, Axel Poschmann: Enabling 3-share Threshold Implementations to any 4-bit S-boxes. Under submission.

7. Hongjun Wu, Phuong Ha Nguyen, Huaxiong Wang, San Ling: Cryptanalysis of the Stream Cipher ZUC in the 3GPP Confidentiality Integrity Algorithms 128-EEA3 128-EIA3. ASIACRYPT2010, rump secession.

PATENT

1. Sebastian Thomas Kutzner, Phuong Ha Nguyen, Axel Poschmann: NIEO TD Ref No: TD/065/12. Title: Construction of Boolean Functions by Lower Degree Boolean Functions, 2012.

In these joint works of this thesis, I played a lead/active role in finding the results. Notably, the result on ZUC is completely based on an observation discovered and presented by Professor Wu Hongjun at *ASIACRYPT* 2010 rump session. Chapter 3 in this thesis about ZUC is completed under guidance of Professor Wu Hongjun.

ABSTRACT

This thesis studies the cryptanalysis and design of some cryptographic algorithms in *symmetric key ciphers*. *Block ciphers* and *Stream ciphers* are two backbone elements in symmetric key ciphers. We cryptanalyze *ZUC* stream cipher, *SERPENT*, *KASUMI* and *SMS4* block ciphers by using some new improved cryptographic methodologies, tools and algorithms. Based on the results obtained in cryptanalysis, we propose some modifications in the designs of *ZUC*, *KASUMI* and *SMS4* to enhance their security. Additionally, we introduce several methodologies to enable and efficiently implement 3-share Threshold Implementation to any 4-bit S-box. Threshold Implementation is widely accepted as a countermeasure against first order Differential Power Analysis in Side Channel Attack and 3-share version is the most efficient in terms of hardware implementation. To support the theoretical results, we construct 3-share Threshold Implementations to S-boxes of *PRESENT*, *SERPENT* ciphers. Notably, 3-share version was believed to be not possible for some S-boxes of *SERPENT* until now.

CONTENTS

1. <i>Introduction</i>	1
1.1 Contributions of this Thesis	2
1.2 Organization of this Thesis	5
2. <i>Background</i>	6
2.1 Block Ciphers	6
2.1.1 Substitution-Permutation Network	7
2.1.2 Feistel Network	9
2.2 Description of Symmetric Key Ciphers analysed in this Thesis	10
2.2.1 SERPENT	10
2.2.2 SMS4	13
2.2.3 KASUMI	16
2.2.4 ZUC	17
2.3 Vectorial Boolean Functions and Computation Tools	17
2.3.1 Notations, Background and Fast Computation Tools	17
2.3.2 Constructing Multidimensional Probability Distribution	20
2.4 Linear Cryptanalysis and Differential Cryptanalysis	21
2.4.1 Linear Cryptanalysis	21
2.4.2 Differential Cryptanalysis	24
2.5 Side Channel Attack	26

<i>Contents</i>	<i>vii</i>
2.5.1 Simple Power Analysis	26
2.5.2 Differential Power Analysis	27
2.5.3 Timing Attack, Cache Timing Attack, Cache Trace Attack . .	28
3. <i>Cryptanalysis of ZUC Stream Cipher</i>	29
3.1 Outline of this Chapter	29
3.2 Notations and short Description of ZUC	31
3.2.1 Notations	31
3.2.2 Description of ZUC	32
3.3 Weaknesses in ZUC Algorithm	34
3.4 The IV_0 Weakness	37
3.4.1 Weak Keys detected by Using IV_0 Weakness	38
3.4.2 The Optimal Checking Set of (iv, iv')	43
3.5 The IV_1 Weakness	45
3.6 Fixed Algorithm of ZUC	53
3.7 Conclusion	53
4. <i>Algorithm 1 of Multidimensional Linear Cryptanalysis</i>	54
4.1 Evolution of Multiple Linear Cryptanalysis	55
4.2 Statistical Model and Algorithm of Hermelin <i>et al.</i>	57
4.2.1 Multidimensional Generalization of Matsui's Alg. 1	58
4.2.2 Algorithm of Hermelin <i>et al.</i> [43].	59
4.3 The Improvements	59
4.3.1 A Formula for N , the Number of Samples Required	59
4.3.2 Analysis of the Multidimensional Alg. 1 in [43]	61
4.3.3 Fast Computation of Empirical Distribution – Method A . . .	63
4.3.4 Fast Computation of Kullback-Leibler Distance – Method B .	65

<i>Contents</i>	<i>viii</i>
4.4 Efficiency of the Improved Algorithm	65
4.5 Application to Cryptanalysis	67
4.6 Conclusions	67
5. <i>Algorithm 2 of Multidimensional Linear Cryptanalysis</i>	69
5.1 Algorithm 2 of Linear Cryptanalysis	69
5.2 Algorithm 2 of Multidimensional Linear Cryptanalysis	72
5.3 Efficient Computation of Distillation Phase of extended Algorithm 2 of Matsui	74
5.3.1 Case 1 and <i>Method 1</i>	75
5.3.2 Case 2 and <i>Method 2</i>	75
5.4 Results on Cryptanalysis of SERPENT	76
5.4.1 Experimental Results on 5-round SERPENT	76
5.4.2 Cryptanalysis of 11-round and 12-round reduced SERPENT	77
5.5 Conclusion	80
6. <i>Related Key Attack on KASUMI Case A5/3</i>	82
6.1 Introduction	82
6.2 KASUMI	83
6.3 Related Key Attacks and KASUMI	88
6.4 Related Key Attacks and A5/3	92
6.4.1 Existing Related Key Attacks	92
6.4.2 Revised Related Key Attacks	94
6.4.3 Implications and Observations	97
6.5 Conclusion	98
7. <i>Improved Differential Cache Trace Analysis on SMS4</i>	100

7.1	Cache Attacks	101
7.2	Differential Cache Trace Attacks	103
7.3	The SMS4 Block Cipher	104
7.4	The Improved Differential Cache Trace Attack on <i>SMS4</i>	104
7.4.1	Step 1: Recovering RK_0	105
7.4.2	Step 2: Recovering RK_1	108
7.4.3	Step 3: Recovering RK_2	110
7.4.4	Step 4: Recovering RK_3	111
7.4.5	The Theoretical Number of Pairs of Samples Needed	111
7.4.6	The Special Attack when $n = 1$	111
7.5	Protection method against the Cache Attacks based on Key-Whitening	112
7.6	Analysis and Experimental Evaluation	114
7.6.1	The Timing Attack Algorithm	114
7.6.2	Application to SMS4	116
7.6.3	Experimental Results	116
7.7	Conclusions	117
8.	<i>Improvements in Threshold Countermeasure</i>	119
8.1	Introduction	119
8.2	The Threshold Implementation Countermeasure	121
8.2.1	Threshold Implementation Countermeasure	121
8.3	One S-box for all	123
8.3.1	The horizontal level	124
8.3.2	The vertical level	126
8.4	Hardware implementation	128
8.4.1	Design flow	128

<i>Contents</i>	<i>x</i>
8.4.2 Serialized architecture	129
8.4.3 Round-based architecture	131
8.4.4 Discussion	131
8.5 Experimental results	134
8.5.1 Measurement setup	134
8.5.2 Side-channel resistance	134
8.6 Conclusion	140
9. <i>Enabling 3-Share Threshold Countermeasure to any 4-bit S-box</i>	141
9.1 Introduction	141
9.2 3-share TIs to cubic 4-bit S-boxes	143
9.3 Decomposition Stage of Threshold Implementation	144
9.3.1 Decomposition of 4-bit S-boxes using a sequential structure . .	144
9.3.2 Decomposition of 4-bit S-boxes using a factorization structure	145
9.3.3 Decomposition of 4-bit S-boxes using a hybrid structure	148
9.4 Sharing stage of Threshold Implementation	150
9.4.1 Sharing stage using a sequential structure	150
9.4.2 Sharing stage using a factorization structure	151
9.5 3-share TIs of S-boxes in B_{16}	154
9.6 Conclusion	160
10. <i>Conclusion</i>	161
<i>Bibliography</i>	163
<i>Appendix</i>	177
A. <i>Cryptanalysis of ZUC stream cipher</i>	178

Contents

xi

<i>B. Algorithm 1 of Multidimensional Linear Cryptanalysis</i>	183
B.1 Proof For Lemma 4.3.1	183
<i>C. The Algorithm 2 of Multidimensional Linear Cryptanalysis</i>	185
C.1 Proof and Description of Lemma 5.3.1	185
C.2 Proof and Description of Lemma 5.3.2	188

1. INTRODUCTION

Cryptography is the study of information security. One of its tasks is providing confidentiality to the transmitted data. To achieve this goal, many cryptographic primitives have been constructed. Among the various topics of cryptographic primitives, *symmetric key primitives* is a very important topic[68]. It has one sub-topic called *symmetric key ciphers* which then contains block ciphers and stream ciphers. These two classes of ciphers play a central role in providing confidentiality. Hence, studying the security of symmetric key ciphers in algorithm, software implementation and hardware implementation aspects is important. A cipher is regarded as secure if there is no attack algorithm, which can reveal the *secret key*, better than *exhaustive key search* algorithm (also known as key brute force) in terms of time complexity. Exhaustive key search is an attack to recover the secret key based on trying all the possible candidate keys. *Cryptanalysis* studies possible attacks on the cipher. Nowadays, the lightweight cryptographic ciphers encourage to seek new design methodologies in order to save hardware implementation costs and power consumption. These problems inspired this thesis to work on cryptanalysis of the following symmetric key ciphers: *ZUC*[7], *SERPENT*[14], *SMS4*[5], *KASUMI*[2], and *PRESENT*[24] and improve the design of *ZUC*, *KASUMI*, *SMS4* and *PRESENT*.

1.1 Contributions of this Thesis

This thesis presents some new and improved methodologies and algorithms for cryptanalysis of *ZUC*, *SERPENT*, *SMS4*, *KASUMI*, as well as some new improvements and correction of design of *ZUC*, *KASUMI*, *SMS4* and *PRESENT*. Our results are summarized as follows:

- In Chapter 3, we cryptanalyze the stream cipher *ZUC*. The cipher *ZUC* is a new stream cipher algorithm and is used as a core cryptosystem in 128-EEA3 & 128-EIA3 of the new 4G mobile standard called LTE (Long Term Evolution) in China. We show that the preliminary *ZUC* is broken based on the observation discovered and presented by Wu Hongjun at the rump session of *ASIACRYPT* 2010[6]. We propose a fixed algorithm which is adopted as the current version of *ZUC*.
- In Chapter 4, we study the Algorithm 1 of the *Multidimensional Linear Cryptanalysis* (MLC)[43] introduced by Hermelin *et.al* in *ACISP* 2008. MLC is an extension of Linear Cryptanalysis discovered by Matsui in *CRYPTO* 1991. We improve the time complexities of two phases of the algorithm by developing two methods. These methods reduce the time complexities from $N2^m$ to $N + 2m2^m$ for phase 1 and 2^{2m} to $3m2^m$ for phase 2. Here m is the number of dimensions and N is the data complexity needed in the algorithm. These methods are based on the *Fast Fourier Transform* (FFT) and *Fast Walsh Hadamard Transform* (FWHT)[27]. We also provide the upper bound for N , i.e., the number of samples needed for the algorithm. Based on this upper bound, we conclude that Hermelin *et.al* formalized the *Multiple Linear Cryptanalysis* of Biryukov *et al.* in *CRYPTO* 2004[21], which is also an extended version of Linear Cryptanalysis.

-
- In Chapter 5, the time complexity improvement in Algorithm 2 of MLC[44] based on FFT and FWHT is presented. If this algorithm is m -dimensional and reveals l bits of the last round key with N plaintext-ciphertext pairs, then its time complexity is reduced from $\mathcal{O}(mN2^l)$ to $\mathcal{O}(N + \lambda 2^{m+l})$ by using our improved methods, where λ is $3(m+l)$ or $4m+3l$. The resulting attacks are the best known key recovery attacks on 11-round and 12-round *SERPENT*. The data, time, and memory complexity of the previously best known attack on 12-round *SERPENT* are reduced by factor of $2^{7.5}$, $2^{11.7}$, and $2^{7.5}$, respectively. This chapter also simulates the experiments of the improved Algorithm 2 of MLC on 5-round *SERPENT*.
 - We introduce the 64-bit key version of block cipher *KASUMI* for algorithm A5/3 by modifying the original one with 128-bit key in Chapter 6. The original KASUMI is a block cipher used in UMTS, GSM, and GPRS mobile communications systems and vulnerable to Related-Key Attack. We prove the upper bound of 3-round related-key differential of 64-bit key version. Based on the result obtained, we conclude that the *sandwich attack*[34], which was used to break the original version in *CRYPTO* 2010, is not applicable to the 64-bit key version. It implies that the modified version resists all kind of related-key attacks.
 - The improved *Differential Cache Trace Attack* (DCTA)[81] on block cipher SMS4 with cache is described in Chapter 7. The DCTA attack was introduced in *CT-RSA* 2011 and the improved attack can break SMS4 cipher with a small pool of chosen plaintexts in few minutes. We show how to construct an efficient algorithm of choosing plaintexts such that the attack becomes applicable to SMS4 ciphers. Moreover, many cryptographic tools are developed to exploit

maximum information provided from the attack model for reducing the number of plaintexts used. We also provide the experimental result of timing attack. A new countermeasure is proposed by modifying the key schedule and is able to apply in general without any trade-off in terms of encryption speed and time. Indeed, this countermeasure uses the *masking* and as we know, this is the first time this idea is used as a countermeasure in this kind of attack.

- In Chapter 8, we focus on *Threshold Implementation* (TI)[73, 74] which is one of the most promising lightweight hardware countermeasures against *first order Differential Power Analysis* (DPA) in *Side-Channel Attack*[53, 54]. We discuss issues towards its applicability and introduce solutions to boost its implementation efficiency. In particular, our contribution is three-fold: first we introduce two methodologies to efficiently implement 3-share TI to a given S-box. Second, as an example, we successfully apply these methodologies to PRESENT and are able to decrease the area requirements of its protected S-box by 37-40%. Third, we present the first successful practical Mutual Information Attack on the original 3-share TI implementation of PRESENT and compare it with a correlation-enhanced collision attack using second-order moments.
- According to [20], 3-share TI can be applied only to 50% of all 4-bit S-boxes so far. In Chapter 9, we study the limitations of decomposition and introduce factorization to enable the 3-share TI to any 4-bit S-box. To highlight the strength of our new methodology, we construct 3-share TIs of S-boxes of SERPENT and 3-share TIs of several 4-bit S-boxes which were believed to be not possible until now .

1.2 Organization of this Thesis

This thesis is organized as follows:

- Chapter 2 provides a background for the latter results. Short introduction of block cipher, Linear Cryptanalysis, Differential Cryptanalysis, brief specifications of *ZUC*, *SERPENT*, *KASUMI*, *SMS4* ciphers, vectorial boolean functions, FFT, FWHT and a short introduction of some attacks in Side-Channel Attack are described.
- In Chapter 3 to Chapter 9, we present all our contributions to cryptanalysis and to the design of particular symmetric key ciphers.
- We conclude the thesis in Chapter 10.

2. BACKGROUND

In this chapter, we revisit some fundamentals of cryptography. The descriptions of the general construction of *SPN* and *Feistel* block ciphers and specifications of *ZUC*, *SERPENT*, *SMS4*, *KASUMI* are recalled. Further, a review of traditional cryptanalysis techniques, i.e., Linear Cryptanalysis, Differential Cryptanalysis and Side Channel Attack is provided. Additionally, the backgrounds of vectorial boolean function, of the fast computation algorithms Fast Fourier Transform and of Fast Walsh Hadamard Transform are presented.

2.1 Block Ciphers

A cipher which uses the same secret key to encrypt and decrypt the transmitted data is called a symmetric key cipher (also known as secret key cipher).

Let $E(\cdot)$ and $D(\cdot)$ denote the encryption and decryption routines of a symmetric key cipher, respectively. Let P , C and k be a *plaintext*, a *ciphertext* and a secret key, respectively. Then the encryption can be defined as $C = E(P, k)$, while decryption as $P = D(C, k)$.

If a symmetric key cipher works on fixed length block of bits (bytes or words), then it is called a block cipher. An input message is divided into many blocks to be operated if it is longer than the block size. All the blocks of a given input message are processed with the same key. A block cipher is called a n -bit block cipher if the block size of plaintext and ciphertext are n bits.

In a stream cipher, each digit of message is encrypted one at a time with the corresponding digit of the keystream, to give a digit of encrypted message. The keystream is a sequence of digits which is generated based on the secret key. In practice, a digit is a bit, a byte or a word. The Figure 2.1 describes an simple encryption of a stream cipher. Note that, we do not intend to provide a definition of stream cipher in this thesis. In this thesis, *ZUC* is the only stream cipher considered and its cryptanalysis is completely based on its algorithm which will be provided in Chapter 3. For more details on general description of a stream cipher, the readers are referred to [68].

Fig. 2.1: Encryption routine of a stream cipher.

Since all studied block ciphers belong to *substitution-permutation network* (SPN) or *Feistel network*, the short introductions of these structures would be provided.

2.1.1 Substitution-Permutation Network

Definition 2.1.1. [68] A substitution-permutation network (SPN) is a cipher composed of a number of stages each involving substitutions and permutations.

The inputs of SPN block ciphers are the plaintext and the secret key. The output (ciphertext) is the result of applying multiple rounds of substitution boxes (S-boxes)

Fig. 2.2: Substitution-permutation network [68].

S and linear permutations P on these inputs. The secret key is used to generate a set of round keys which are introduced in each round of the encryption (or decryption) routine. The decryption is the reverse process of the encryption.

In a cipher, a nonlinear component called S-box and the linear permutation P are used for fulfilling the *confusion property* and the *diffusion property*, respectively. The aim of confusion property is that every bit of S-box output depends on many bits of S-box input. In practice, a good S-box always has such diffusion property. The linear permutation layer P distributes the output bits of any S-box to as many S-box inputs in the next round as possible. Hence, doing a number of rounds, changing any input bit (in plaintext or secret key) will change the ciphertext completely, in a pseudorandom manner. These two properties are required to construct a good cipher defined by Claude Shannon [86]. According to Shannon, if a cipher has these two properties, then all bits of ciphertext will be obtained by mixing the different bits of the plaintext and the secret key in a complex manner. Therefore, the secret key can

not be recovered even if an adversary can collect many pairs of plaintext-ciphertext produced by this key.

Definition 2.1.2. [68] An *iterated block cipher* is a block cipher involving the sequential repetition of an internal function called a round function. Parameters include the number of rounds r , the block size n (in bits), and the key size k (in bits) of the secret key K from which r subkeys K_i (round keys) are derived. For invertibility (allowing unique decryption), for each value K_i the round function is a bijection on the round input.

Usually, an SPN block cipher is an iterated block cipher.

2.1.2 Feistel Network

Feistel network was introduced by cryptographer Horst Feistel. This construction is used in many block ciphers.

Definition 2.1.3. [68] A Feistel cipher is an iterated cipher mapping a $2t$ -bit plaintext (L_0, R_0) , for t -bit blocks L_0 and R_0 , to a ciphertext (R_r, L_r) , through an r -round process where $r \geq 1$. For $1 \leq i \leq r$, round i maps $(L_{i-1}, R_{i-1}) \xrightarrow{K_i} (L_i, R_i)$ as follows: $L_i = R_{i-1}$, $R_i = L_{i-1} \oplus f(R_{i-1}, L_i)$, where each subkey (or called round key) K_i is derived from secret key k and f is a boolean function.

The function f is not required to be invertible. Figure 2.3 describes a Feistel structure.

Notably, there are many ciphers which are constructed based on Feistel structure but their structures are modified from the original one, for example SMS4[98], KASUMI[1] etc. It means there are many types of Feistel structures nowadays.

Fig. 2.3: A Feistel structure.

2.2 Description of Symmetric Key Ciphers analysed in this Thesis

In this section, a review of the cryptanalyzed block ciphers, i.e., *SERPENT*, *SMS4*, *KASUMI*, and stream cipher *ZUC* are provided.

2.2.1 *SERPENT*

SERPENT is a block cipher designed by Ross Anderson, Eli Biham, and Lars Knudsen and first introduced at *FSE* 1998 [14] (as known as *SERPENT* – 0). Then, it was submitted to Advanced Encryption Standard (AES) contest and chosen as one of 5 finalists with some modifications in key schedule (called *SERPENT* – 1). It has a 128-bit block size and supports secret key lengths of 128, 192 and 256 bits. The input and output of *SERPENT* are a 128-bit plaintext P and a 128-bit ciphertext C . *SERPENT* is a 32-round SPN cipher and operates on four 32-bit

words. In each round, one of the eight 4-bit S-boxes is chosen and is used 32 times in parallel. The design of *SERPENT* allows all operations to be processed in parallel, using thirty two 4-bit slices. Hence, the parallelism of execution is maximized. It means that the speed of encryption is enhanced.

The Key Schedule and Decryption Routine

The aim of the key schedule is to generate thirty three 128-bit round keys $K_0, \dots, K_{32}$ from the secret key. The input of the key schedule is 128-, 192-, or 256-bit secret key. For more details on the key schedule and decryption routine, we refer the reader to [14] because this thesis is not focused on the key schedule and decryption routine.

The Encryption Routine

For the sake of convenience, we use the notation in [16].

The indices of the rounds are counted from 0 to 31. Let $\hat{B}_i$ denote intermediate value of round i and it is a concatenation of four 32-bit word (X_3, X_2, X_1, X_0) , where X_0 is the least significant word and X_3 is the most significant word. *SERPENT* has eight 4-bit S-boxes $S_0(x), \dots, S_7(x)$. In each round, the substitution layer (S-box layer) $\hat{S}_i$ is the concatenation of thirty two $S_{j=i \bmod 8}$ S-boxes. The input of the S-box layer $(\hat{B}_i \oplus K_i)$ is divided into 32 nibbles which are feeded into thirty two S-boxes S_j in round i . The output of S-box layer goes through the linear transformation layer L before becoming the input of the next round, i.e., round $(i + 1)$. Note that in the last round, the S-box S_7 is chosen in S-boxes layer. The encryption routine can be described as follows:

$$\begin{aligned}
\hat{B}_0 &= IP(P), \\
\hat{B}_{i+1} &= R_i(\hat{B}_i), \\
C &= FP(\hat{B}_{32}),
\end{aligned} \tag{2.1}$$

where IP and FP are the initial permutation and final permutation which were described in [94], respectively.

$$\begin{aligned}
R_i(X) &= L(\hat{S}_i(X \oplus K_i)), \text{ } i=0, \dots, 30 \\
R_{31} &= \hat{S}_7(X \oplus K_{31}) \oplus K_{32} \text{ } i=31,
\end{aligned} \tag{2.2}$$

and linear transformation L :

$$\begin{aligned}
(X_3, X_2, X_1, X_0) &= \hat{S}_i(\hat{B}_i \oplus K_i), \\
X_0 &= X_0 \lll 13, \\
X_2 &= X_2 \lll 3, \\
X_1 &= X_1 \oplus X_0 \oplus X_2,
\end{aligned}$$

$$\begin{aligned}
X_3 &= X_3 \oplus X_2 \oplus (X_0 \lll 3), \\
X_1 &= X_1 \lll 1, \\
X_3 &= X_3 \lll 7, \\
X_0 &= X_1 \oplus X_0 \oplus X_3, \\
X_2 &= X_3 \oplus X_2 \oplus (X_1 \ll 7), \\
X_0 &= X_0 \lll 5, \\
X_2 &= X_2 \lll 22, \\
\hat{B}_{i+1} &= (X_3, X_2, X_1, X_0),
\end{aligned}$$

where $\lll$ and $\ll$ denote rotation and shift operators, respectively.

This is the list of *SERPENT* S-boxes.

$$\begin{aligned}
S_0 &= [3, 8, 15, 1, 10, 6, 5, 11, 14, 13, 4, 2, 7, 0, 9, 12], \\
S_1 &= [15, 12, 2, 7, 9, 0, 5, 10, 1, 11, 14, 8, 6, 13, 3, 4], \\
S_2 &= [8, 6, 7, 9, 3, 12, 10, 15, 13, 1, 14, 4, 0, 11, 5, 2], \\
S_3 &= [0, 15, 11, 8, 12, 9, 6, 3, 13, 1, 2, 4, 10, 7, 5, 14], \\
S_4 &= [1, 15, 8, 3, 12, 0, 11, 6, 2, 5, 4, 10, 9, 14, 7, 13], \\
S_5 &= [15, 5, 2, 11, 4, 10, 9, 12, 0, 3, 14, 8, 13, 6, 7, 1], \\
S_6 &= [7, 2, 12, 5, 8, 4, 6, 11, 14, 9, 1, 15, 13, 3, 10, 0], \\
S_7 &= [1, 13, 15, 0, 14, 8, 2, 11, 7, 4, 12, 10, 9, 3, 5, 6].
\end{aligned} \tag{2.3}$$

2.2.2 SMS4

SMS4 is a 32-round *Feistel* block cipher with 128-bit block size and 128-bit key size. It is used in the Chinese National Standard for Wireless LAN WAPI (Wired

Authentication and Privacy Infrastructure). The *SMS4* algorithm was introduced by Prof. LU Shu-wang in January, 2006. The input and output of *SMS4* are a 128-bit plaintext P and a 128-bit ciphertext C , respectively. The encryption and decryption routines are similar to each other except for the order of round keys, i.e., they are used in the reverse order. For the sake of convenience, we adopted the notations used in [98].

Key Schedule and Decryption Routine

Key schedule produces thirty two 32-bit round keys $RK_0, \dots, RK_{31}$ based on the given secret key. The reader is referred to [5] for full descriptions of key schedule and decryption routine. One important fact of the key schedule is that if any four consecutive round keys $RK_i, \dots, RK_{i+3}$, $0 \leq i \leq 28$, are known, then the secret key can be fully recovered.

S-box $S(x)$

An 8-bit S-box $S(\cdot)$ is used in *SMS4*. Let $x = (x_0 || x_1)$, $y = (y_0 || y_1)$ denote 8-bit input and 8-bit output of $S(\cdot)$, respectively, where x_0, x_1, y_0, y_1 are 4-bit nibbles. Then, the output $y = (y_0 || y_1) = S(x) = S(x_0 || x_1)$ is presented in Figure 2.4.

In that figure, all the numbers are written in hexadecimal. The index x_0 points the row and index x_1 refers to the column. In order to remove any ambiguity, let us take a look at the following example: $S(ef) = 84$.

Encryption Routine

The cipher has 32-rounds, with each round comprising of an Feistel structure as shown in Figure 2.5. The inputs to round i ($0 \leq i \leq 31$) are four 32 bit words X_i ,

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	d6	90	e9	fe	cc	e1	3d	b7	16	b6	14	c2	28	fb	2c	05
1	2b	67	9a	76	2a	be	04	c3	aa	44	13	26	49	86	06	99
2	9c	42	50	f4	91	ef	98	7a	33	54	0b	43	ed	cf	ac	62
3	e4	b3	1c	a9	c9	08	e8	95	80	df	94	fa	75	8f	3f	a6
4	47	07	a7	fc	f3	73	17	ba	83	59	3c	19	e6	85	4f	a8
5	68	6b	81	b2	71	64	da	8b	f8	eb	0f	4b	70	56	9d	35
6	1e	24	0e	5e	63	58	d1	a2	25	22	7c	3b	01	21	78	87
7	d4	00	46	57	9f	d3	27	52	4c	36	02	e7	a0	c4	c8	9e
8	ea	bf	8a	d2	40	c7	38	b5	a3	f7	f2	ce	f9	61	15	a1
9	e0	ae	5d	a4	9b	34	1a	55	ad	93	32	30	f5	8c	b1	e3
a	1d	f6	e2	2e	82	66	ca	60	c0	29	23	ab	0d	53	4e	6f
b	d5	db	37	45	de	fd	8e	2f	03	ff	6a	72	6d	6c	5b	51
c	8d	1b	af	92	bb	dd	bc	7f	11	d9	5c	41	1f	10	5a	d8
d	0a	c1	31	88	a5	cd	7b	bd	2d	74	d0	12	b8	e5	b4	b0
e	89	69	97	4a	0c	96	77	7e	65	b9	f1	09	c5	6e	c6	84
f	18	f0	7d	ec	3a	dc	4d	20	79	ee	5f	3e	d7	cb	39	48

Fig. 2.4: SMS4 S-box $S(\cdot)$ [5].

X_{i+1} , X_{i+2} , and X_{i+3} . A round of *SMS4* can be written as follows [97]:

$$\begin{aligned}
 X_{i+4} &= F(X_i, X_{i+1}, X_{i+2}, X_{i+3}) \\
 &= X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus RK_i),
 \end{aligned} \tag{2.4}$$

where RK_i ($0 \leq i \leq 31$) is a 32 bit round key and T is a function comprising of a layer of substitution followed by a diffusion. The 32-bit inputs X_i and round key RK_i ($0 \leq i \leq 31$) are divided into 4 parts, each of 8 bits. These are respectively denoted $X_{i,0}, X_{i,1}, X_{i,2}, X_{i,3}$ and $RK_{i,0}, RK_{i,1}, RK_{i,2}, RK_{i,3}$.

Substitution is achieved by four identical 8×8 S-boxes denoted $S_3(\cdot), S_2(\cdot), S_1(\cdot)$, and $S_0(\cdot)$. The input to the S-box S_j , where $0 \leq j \leq 3$ is $(X_{i,j} \oplus X_{i+1,j} \oplus X_{i+2,j} \oplus X_{i+3,j} \oplus RK_{i,j})$.

Diffusion is provided by an L function defined as follows.

$$\begin{aligned}
 C &= L(B) \\
 &= B \oplus (B \lll 2) \oplus (B \lll 10) \oplus (B \lll 18) \oplus (B \lll 24),
 \end{aligned} \tag{2.5}$$

where $\lll i$ represents left rotation by i bits and B is the output of the substitution layer, i.e., $B = (S_3(\cdot)_{(7\dots 0)} | S_2(\cdot)_{(7\dots 0)} | S_1(\cdot)_{(7\dots 0)} | S_0(\cdot)_{(7\dots 0)})$. Similarly,

$$\begin{aligned} B \lll 2 &= (S_3(\cdot)_{(5\dots 0)} | S_2(\cdot)_{(7\dots 0)} | S_1(\cdot)_{(7\dots 0)} | S_0(\cdot)_{(7\dots 0)} | S_3(\cdot)_{(7,6)}), \\ B \lll 10 &= (S_2(\cdot)_{(5\dots 0)} | S_1(\cdot)_{(7\dots 0)} | S_0(\cdot)_{(7\dots 0)} | S_3(\cdot)_{(7\dots 0)} | S_2(\cdot)_{(7,6)}), \\ B \lll 18 &= (S_1(\cdot)_{(5\dots 0)} | S_0(\cdot)_{(7\dots 0)} | S_3(\cdot)_{(7\dots 0)} | S_2(\cdot)_{(7\dots 0)} | S_1(\cdot)_{(7,6)}), \\ B \lll 24 &= (S_0(\cdot)_{(7\dots 0)} | S_3(\cdot)_{(7\dots 0)} | S_2(\cdot)_{(7\dots 0)} | S_1(\cdot)_{(7\dots 0)}). \end{aligned} \quad (2.6)$$

Fig. 2.5: Description of round i in SMS4 [97].

2.2.3 KASUMI

KASUMI is a 128-bit key size block cipher of *Feistel* network. It has 64-bit block size and is used in UMTS, GSM, and GPRS mobile communications systems. In the case of UMTS, *KASUMI* provides the confidentiality (UEA1) (f8) and integrity algorithms (f9) (UIA1) [1]. For GSM, *KASUMI* is used in the A5/3 key stream generator whereas in GPRS, it is utilised in the GEA3 key stream generator.

KASUMI was developed from MISTY1 for 3GPP. It is used in UMTS security system by the Security Algorithms Group of Experts (SAGE), a part of the European standards body ETSI.

The description of *KASUMI* will be presented in detail in Chapter 6.

2.2.4 ZUC

The cryptanalysis of *ZUC* stream cipher is based on the weakness in its structure. Therefore, the description of *ZUC* will be explained in details in Chapter 3.

2.3 Vectorial Boolean Functions and Computation Tools

In this section, we introduce all necessary notations and background which are used in Chapters 4 and 5 of Multidimensional Linear Cryptanalysis.

2.3.1 Notations, Background and Fast Computation Tools

We follow the notations used in [43]. Let V_m be the space of m -dimensional binary vectors, i.e., $V_m := GF(2)^m$. The inner product of 2 vectors $a = (a_1, \dots, a_m), b = (b_1, \dots, b_m), a, b \in V_m$ is $ab = \bigoplus_{i=1}^m a_i b_i$.

The function $f : V_m \rightarrow V_1$ is called a Boolean function and $f : V_n \rightarrow V_m, f = (f_1, \dots, f_m)$ is called a vectorial Boolean function, where each f_i is a Boolean function for all $i = 1, \dots, m$.

Let X be a random variable in V_m . Let $p_\eta = Pr(X = \eta)$, with $\eta \in V_m$. Then $p = (p_0, p_1, \dots, p_{2^m-1})$ is the probability distribution (p.d) of random variable X . If we associate with a vectorial Boolean function $f : V_n \rightarrow V_m$ an random variable $Y := f(X)$, where X is uniformly distributed in V_n , then the p.d. of Y is $p(f) := (p_0(f), \dots, p_{2^m-1}(f))$ where $p_\eta(f) = Pr(f(X) = \eta)$, for all $\eta \in V_m$. Two Boolean functions f and g are called statistically independent if their associated random variable's $f(X)$ and $g(Y)$ are statistically independent, with X, Y uniform in V_n .

Let ϵ denote a bias of the random variable X which is computed as follows: $\epsilon = Pr(X = 0) - 1/2$. The correlation between a binary random variable X and 0 is $\rho = Pr(X = 0) - Pr(X = 1) = 2\epsilon$. Let $g : V_m \rightarrow V_1$ be a Boolean function. Its correlation with 0 is defined as

$$\rho = 2^{-m}(\#\{\eta \in V_m | g(\eta) = 0\} - \#\{\eta \in V_m | g(\eta) = 1\}) = 2Pr(g(X) = 0) - 1,$$

where X is uniformly distributed in V_m .

Definition 2.3.1. Let $p = (p_0, \dots, p_M)$ and $q = (q_0, \dots, q_M)$ be two p.d.'s. Then their (mutual) capacity is defined as

$$C(p||q) = \sum_{\eta=0}^M \frac{(p_\eta - q_\eta)^2}{q_\eta}. \quad (2.7)$$

Definition 2.3.2. The relative entropy or the Kullback-Leibler (KL) distance between two distributions $p = (p_0, \dots, p_M)$ and $q = (q_0, \dots, q_M)$ is defined as

$$D(q||p) = \sum_{\eta=0}^M q_\eta \log \frac{q_\eta}{p_\eta}.$$

In [27], Collard *et al.* presented the following theorems concerning circulant matrices.

Theorem 2.3.3. A circulant $\mathbf{S}$ of level k and type $(m, n, o, \dots, r)$ is diagonalizable by the unitary matrix $\mathbf{F} = \mathbf{F}_m \otimes \mathbf{F}_n \otimes \mathbf{F}_o \otimes \dots \otimes \mathbf{F}_r$

$$\mathbf{S} = \mathbf{F}^* \text{diag}(\lambda) \mathbf{F},$$

where λ is the vector of eigenvalues of $\mathbf{S}$, the symbol $\otimes$ is the Kronecker product and

$\mathbf{F}_n$ is the Fourier matrix of size $n \times n$ defined by:

$$\mathbf{F}_n(i, j) = \frac{1}{\sqrt{n}} \omega^{ij}, \quad (0 \leq i, j \leq n-1)$$

with

$$\omega = e^{\frac{2\pi\sqrt{-1}}{n}}.$$

Theorem 2.3.4. *The eigenvalues vector λ of a circulant matrix $\mathbf{S}$ of level k and type $(m, n, o, \dots, r)$ can be computed with the following matrix-vector product:*

$$\lambda = \mathbf{F}\mathbf{S}(:, 1)\sqrt{mno\dots r},$$

where $\mathbf{S}(:, 1)$ means we take the first column of $\mathbf{S}$.

We recall important results on the Fast Fourier Transform [30], Fast Walsh-Hadamard Transform [95] and Parseval's theorem. Given an m -dimensional vector $\mathbf{E} = (E_1, \dots, E_m)$ and a matrix $\mathbf{F}^{m \times m}$, we have m -dimensional vector

$$\mathbf{D} = \mathbf{F}\mathbf{E}^T,$$

where $\mathbf{E}^T$ is the transpose of $\mathbf{E}$ and $\mathbf{F}$ is a Hadamard matrix if $\mathbf{F}(i, j) = (-1)^{ij}$, for all $i, j = 0, \dots, m-1$. If matrix $\mathbf{F}$ is either Fourier or Hadamard, vector $\mathbf{D}$ can be computed with complexity $\mathcal{O}(m \log m)$ instead of $\mathcal{O}(m^2)$ by Fast Fourier Transform or Fast Walsh-Hadamard Transform, respectively. We recall basic facts due to Parseval:

Let $f : V_m \rightarrow \mathcal{R}$ where $\mathcal{R}$ is the real field, and $a \in V_m$. We define $f_1(a) =$

$\sum_{b \in V_m} (-1)^{ab} f(b)$, $A = \sum_{a \in V_m} f^2(a)$ and $A_1 = \sum_{a \in V_m} f_1^2(a)$. Then

$$2^m A = A_1$$

or

$$2^m \left(\sum_{a \in V_m} f^2(a) \right) = \sum_{a \in V_m} f_1^2(a). \quad (2.8)$$

2.3.2 Constructing Multidimensional Probability Distribution

Let $f : V_l \rightarrow V_n$ be a vectorial Boolean function and binary vectors $w_i \in V_n, u_i \in V_l, i = 1, \dots, m$ be selection patterns such that pairs of input and output masks (u_i, w_i) are linearly independent. Define the functions g_i as

$$g_i(\eta) = w_i f(\eta) \oplus u_i \eta, \quad \forall \eta \in V_l, i = 1, \dots, m$$

and g_i has correlation $\rho_i, i = 1, \dots, m$. Then $\rho_1, \dots, \rho_m$ are called the base-correlations, and $g_1, \dots, g_m$ are the base approximations of f . Let $g = (g_1, \dots, g_m)$ be an m -dimensional vectorial Boolean function, and matrices $W = (w_1, \dots, w_m)$ and $U = (u_1, \dots, u_m)$ contain the output and input masks for each of the g_i , then we find the p.d. $p = (p_0, \dots, p_{2^m-1})$ of

$$g(\eta) = W f(\eta) \oplus U \eta.$$

Lemma 2.3.5. [43] Let $g = (g_1, \dots, g_m) : V_l \rightarrow V_m$ be a vectorial Boolean function and $p = (p_0, \dots, p_{2^m-1})$ its p.d. Then

$$2^l p_\eta = 2^{-m} \sum_{a \in V_m} \sum_{b \in V_l} (-1)^{a(g(b) \oplus \eta)}.$$

Define

$$\rho(a) = 2^{-l} \sum_{b \in V_l} (-1)^{ag(b)} = Pr(ag(X) = 0) - Pr(ag(X) = 1),$$

where X is an random variable uniformly distributed in V_l .

Corollary 2.3.6. *Let $g : V_n \rightarrow V_m$ be a Boolean function with p.d. p and correlations $\rho(a)$ of the combined approximations ag , for all $a \in V_m$. Then for $\eta \in V_m$,*

$$p_\eta = 2^{-m} \sum_{a \in V_m} (-1)^{a\eta} \rho(a). \quad (2.9)$$

2.4 Linear Cryptanalysis and Differential Cryptanalysis

The aim of cryptanalysis is to analyze the security level achieved by the cipher. There are many advanced cryptanalytic tools developed for this purpose. In this thesis, we only focus on 2 types of advanced cryptanalytic tools, i.e., *Linear Cryptanalysis* and *Differential Cryptanalysis*

2.4.1 Linear Cryptanalysis

Linear Cryptanalysis (LC) was introduced by M.Matsui and Yamagishi in 1992 [67]. Subsequently, the block cipher Data Encryption Standard (DES) was reported as being broken at CRYPTO 1992 [65]. In LC, there are two algorithms, i.e., Algorithm 1 and Algorithm 2. Indeed, the Algorithm 2 is developed directly from Algorithm 1. From then onwards, LC is accepted as one of the most powerful cryptanalytic tools and is used in cryptanalyzing many ciphers. One of the requirements for the design of a cipher is that the cipher has to resist LC.

The results of Multidimensional Linear Cryptanalysis (*MLC*), which is an ex-

tension of LC, is improved in this thesis. For the sake of convenience, we follow the notations in [72].

Algorithm 1

LC exploits the linear relationship g between the plaintext P , the ciphertext C and extended key K constructed from secret key k , over the cipher $E_k(\cdot)$. The relationship g is formally written as follows:

$$\begin{aligned} g &:= uP \oplus vC \oplus wK, \\ Pr(g = 0) &= 1/2 + \epsilon. \end{aligned} \tag{2.10}$$

where u, v and w are selection patterns (or masks) for plaintext, ciphertext and extend key, respectively. The ϵ is called bias of linear approximation g . The reader is referred to [65] for more details on computing ϵ over cipher $E_k(\cdot)$ by using *Piling Up Lemma*.

The parity bit value wK can be recovered if sufficient number of random plaintext-ciphertext (P, C) pairs N is provided. Indeed, the parameter N is proportional to c/ϵ^2 , where c is a constant which depends on the cipher $E_k(\cdot)$ and the aim of the adversary. Denote N^+ to be the number of (P, C) pairs such that the value $(uP \oplus vC) > 0$ and assume that $\epsilon > 0$. If $N^+/N > 1/2$, then $wK = 0$ else $wK = 1$.

The parameters of the Algorithm 1 are as follows:

- Time Complexity : N .
- Sample Complexity: N .
- Number of recovered bits: 1.

Algorithm 2

Algorithm 1 can recover only 1 bit of key information, i.e., the parity bit wK . Algorithm 2 is developed to recover multiple bits (l bits). Assume that the cipher $E_k(\cdot)$ is r -round SPN block cipher and a linear approximation over $r - 1$ rounds $(uP \oplus vX \oplus wK)$ is constructed with bias ϵ , where X is the output of the $(r - 1)$ -th round. In cipher $E_k(\cdot)$, the value vX can be replaced by a function which involves l bits of last round key (denoted by K_l) and of the ciphertext C (denoted by C_l). This relation is formalized as follows: $uX = f(K_l \oplus C_l)$, where $f(K_l \oplus C_l) = zS^{-1}(K_l \oplus C_l)$, S^{-1} denotes the inverse S-boxes layer, z is the section pattern corresponding to v on the input of the last round. In fact, z depends on the linear transformation layer, selection pattern v and $f(K_l \oplus C_l)$. Since the approximation $(uP \oplus vX \oplus wK)$ has a bias ϵ , the new following r -round relation/linear approximation has ϵ as well, i.e.,

$$\begin{aligned} g &= uP \oplus f(K_l \oplus C_l), \\ Pr(g = 0) &= 1/2 + \epsilon. \end{aligned} \tag{2.11}$$

Since K_l has l bits, there are 2^l key candidates of K_l . The correct key is recovered based on the *Wrong Key Hypothesis* [42]. The Wrong Key Hypothesis assumes that if a wrong key is substituted into g , then g is uniformly distributed. If the chosen key is correct, then g is not uniformly distributed. So, all 2^l candidates are checked with N pairs of (P, C) . The key which has maximal absolute value $|N^+/N - 1/2|$ is chosen as the correct key.

The parameters of the Algorithm 2 are as follows:

- Time Complexity : $2^l N$.
- Sample Complexity: N .

- Number of recovered bits: l .

Many papers were published to improve the two Matsui's Algorithms. One direction is to try reducing the number N by using multiple linear approximations instead of single one [21, 25, 43, 49]. Another direction is to decrease the time complexity [27, 66].

2.4.2 Differential Cryptanalysis

Differential Cryptanalysis was proposed by E.Biham and A.Shamir for cryptanalyzing *DES*-like cipher [18, 19] which was already known by *NSA* (known as "T-attack" or "Tickle attack") [59] as early as 1974. It becomes one of the most powerful tools in cryptanalyzing block ciphers and hash functions.

The method works based on the following fact: For an n -bit S-box S , there exists an input difference α and an output difference β such that $Pr[\alpha \rightarrow \beta] = p > 0$. The $\alpha, \beta, Pr[\alpha \rightarrow \beta]$ is defined as follows:

- Denote (X, X') a pair of input of S-box S .
- Define $\alpha = X \oplus X'$ and $\beta = S(X) \oplus S(X')$.
- Denote $N(\alpha, \beta) = \#\{ \text{pair } (X, X') : \text{if } X \oplus X' = \alpha, \text{ then } S(X) \oplus S(X') = \beta \}$.
The total number of pairs (X, X') such that $X \oplus X' = \alpha$ is 2^n .
- The value $p = Pr[\alpha \rightarrow \beta]$ is equal to $N(\alpha, \beta)/2^n$, i.e., $Pr[\alpha \rightarrow \beta] = N(\alpha, \beta)/2^n$.

Assume that considered cipher $E_k(\cdot)$ has r rounds of SPN structure with n -bit plaintext P and n -bit ciphertext C . A part of last round key (let denote by K_l with l -bit wide) of cipher $E_k(\cdot)$ can be recovered if there exists a pair of difference input and output (α, β) (or called differential) of $(r - 1)$ rounds such that the

probability $p = Pr[\alpha \rightarrow \beta] > 1/2^n$ (called non-negligible differential). The output of round $(r - 1)$ can be computed based on the ciphertext C , the key K_l , the inverse function of S-boxes layer and linear transformation layer. For each key candidates $\bar{k}$ of K_l , we compute the number $N(\alpha, \beta, \bar{k})$. The key candidate $\bar{k}$ which has the largest $N(\alpha, \beta, \bar{k})$ is chosen as the correct key by following the *Wrong Key Hypothesis* [18]. It is very difficult to compute the probability of a differential (α, β) . Instead of computing the probability of a differential, the probability of a *differential characteristic* of (α, β) is computed. The differential characteristic is a trail by which the difference input α propagates so the output difference is β . We refer the reader to [46] for more details on the construction of differential characteristic of a given cipher.

Related key Differential Attack

In principle, the related key differential cryptanalysis is similar to traditional differential cryptanalysis, except that for the former case, the adversary can introduce the difference not only in the plaintext but also in the key.

Definition 2.4.1. [52] We say that a related key differential $\alpha \rightarrow \beta$ with the key difference Δk holds for cipher E with probability p , if

$$\Pr_{P,k}[E_k(P) \oplus E_{k \oplus \Delta k}(P \oplus \alpha) = \beta] = p,$$

where $E_k(\cdot)$ denotes encryption through E with the key k and P is a plaintext. For the ease of exposition, we denote this event by $Pr(\alpha \xrightarrow[\Delta k]{E} \beta)$.

2.5 Side Channel Attack

Side Channel Attack (SAC) introduced by Kocher *et al.* in CRYPTO 1999 [54] is the most powerful attack on implementations of cryptosystem. A SAC is any attack which uses side information observed on physical implementation of a cryptographic system rather than cryptanalysis of weakness in the algorithms. The side information can be time information, power consumption, electromagnetic or the sound and all of them provide the adversary extra information on the cryptosystem to break the cipher.

This thesis analyses the power consumption of an cryptosystem implementation and information that is leaked about the secret key. This type of attack is called *Power Analysis*. The power consumption describes exactly how data is processed and provides clues about its performance on the operation. It is divided into 2 main categories *Simple Power Analysis* (SPA) and *Differential Power Analysis* (DPA).

2.5.1 Simple Power Analysis

In SPA, the power consumption of the cryptographic hardware device is recorded when a data is in process, i.e., usually when encrypting a plaintext. A record of power consumption is called *power trace*. For example, a power trace of 2-round encryption of AES cipher is illustrated in Figure 2.6.

The single-shot SPA attacks use only one trace to recover the key. Otherwise, the multiple-shot SPA attacks have to appeal many traces. In SPA attack, the hardware device under attack is required to be carefully templated, i.e., the adversary have a wide access to analyze how the power consumption depends on the data in process and on the operation it performs. The power consumption strongly relies on processed data in the hardware device.

Fig. 2.6: The Power Trace of 2-round encryption of AES [61].

2.5.2 Differential Power Analysis

DPA is one kind of Power Analysis Attack which utilizes a large number of power traces to analyze the power consumption at a fixed moment of time. The difference of DPA from SPA is that DPA does not need to characterize the attacked device. We do not study DPA in this thesis so that the reader is referred to [61] for more details.

To prevent the DPA, the *masking* countermeasure is proposed. The aim of masking method is to make the power consumption independent from data by masking the data with a random variable. The power traces are totally different from each other because the mask is changed for each encryption. But the presence of *glitch* still makes the masked cipher vulnerable to DPA attack [62, 63].

Definition 2.5.1. [61] The different arrival times of input signals at a combinational cell lead to temporary states of the output of this cell. Such temporary states of the output of a combinational cell are called *glitches* or *dynamic hazards*. Glitches in CMOS circuits are data dependent and have a strong impact on the dynamic power consumption.

To enhance a masked cipher, the technique of Multi-Party Computation is applied [73–75]. This method is called *Threshold Implementation* (TI). In [80], a pro-

tected S-boxes of *PRESENT* block cipher [24] is efficiently implemented with TI method to resist against SAC. We provide a deep insight into hardware implementation with TI protection of all 4-bit S-boxes and propose a new efficient methodology in constructing protected S-boxes in the latter chapters of this thesis.

2.5.3 Timing Attack, Cache Timing Attack, Cache Trace Attack

A *Timing Attack* is an attack which exploits the relation between the data and operation on which the data performs. In a cryptographic algorithm, the execution time of an operation is very different for each value of the processed data. Hence, the knowledge of the execution time reveals the value of processed data [53].

The *Cache Timing Attack* is an attack based on measuring the execution time of operations, which strongly depends on data in the device having a cache. The cache is used to enhance the speed of encryption (or decryption) process so that there is a time variation between the data which is in cache and not in cache [11]. In cache timing attack, adversary only knows the time of an encryption. The *Cache Trace Attack* is an attack on cryptosystem using cache. In this attack, the power traces of encryptions are used. The power trace reveals much side information of an encryption, i.e., how the data and operations are processed through the encryption. Hence, the cache trace attack is more powerful than cache timing attack. The *Differential Cache Trace Attack* (DCTA) is an attack on the cache which is very efficient on *Feistel* network block cipher [81, 82]. The DCTA exploits the Feistel structure of cipher with a cache to improve not only cache trace attack but also cache timing attack. We will study this attack in Chapter 7.

3. CRYPTANALYSIS OF ZUC STREAM CIPHER

ZUC is a 128-bit key stream cipher and is proposed to be the standard of GSM in China. Two weaknesses which lead to two attacks in *ZUC* algorithm is studied in this chapter. In the first attack, the secret key k can be recovered with the success probability of $2^{-15.2}$ and the practical sample complexity of $2^{23.5}$. The second attack is able to retrieve any key k with the time complexity of 2^{100} and sample complexity of 2^{62} . Based on the results received, we conclude that *ZUC* stream cipher is broken and we propose a fix that removes the weakness. The main important observations in this Chapter come from Prof Wu Hongjun. In this chapter, all the algorithms are developed under guidance of Prof Wu Hongjun.

3.1 *Outline of this Chapter*

ZUC stream cipher was designed by the Data Assurance and Communication Security Research Center of the Chinese Academy of Sciences with 128-bit key in 2010. The cipher is used in the Long Term Evolution (LTE)- the 4G mobile standard in China. The *ZUC* cipher plays a central role in the LTE encryption algorithm called 128-EEA3 and the LTE integrity algorithm called 128-EIA3. We show that the version v.1-5 of *ZUC* [7] is broken based on an observation discovered and presented by Wu in ASIACRYPT2010 rump session [6]. Wu observed that for a certain secret key k there exists a pair (iv, iv') , $iv \neq iv'$, such that the key stream generated from (iv, k) is similar to that from (iv', k) (called collision). Based on the collision, the

secret key is recovered. After studying *ZUC* algorithm, we found two weaknesses in the algorithm. The first weakness makes the first attack recover $2^{112.8}$ out of 2^{128} keys (called weak keys) with $2^{23.5}$ (iv, iv') pairs (called optimal set of iv) and time complexity of 2^{106} . In the second attack, the second weakness can be used to recover any secret key k with 2^{62} (iv, iv') pairs and time complexity of 2^{100} . Actually, if the key is a weak key then the combined attack of attack 1 and attack 2 can reduce the time complexity from 2^{100} to 2^{70} but the number of samples needed is increased from $2^{23.5}$ to 2^{62} . Several optimal tools based on the *divide-and-conquer* are developed to achieve the optimal results in terms of time complexity.

The chapter is organized as follows. In 3.2, the description of the *ZUC* algorithm is provided. We point out weaknesses in *ZUC* algorithm in 3.3. The first and the second attack are presented in 3.4 and in 3.5, respectively. Subsequently, the fixed algorithm is introduced in 3.6. Then we conclude the chapter in 3.7.

3.2 Notations and short Description of ZUC

3.2.1 Notations

In this chapter, we follow the notations used in [7] .

$+$	The addition of two integers
$=$	The assignment operator
$\oplus$	The bit-wise exclusive-or operation of integers
mod	The modulo operation of integers
ab	The product of integers a and b
$a b$	The concatenation of string a and b
$a \lll k$	The k -bit cyclic shift of a to the left
$a \ggg k$	The k -bit cyclic shift of a to the right
$a \gg 1$	The 1-bit right shift of integer a
a_H	The leftmost 16 bits of integer a
a_L	The rightmost 16 bits of integer a
$(a_1, a_2, \dots, a_n) \rightarrow (b_1, b_2, \dots, b_n)$	It assigns the values of a_i to b_i in parallel
0_n	The sequel of n bits 0
1_n	The sequel of n bits 1
$\bar{y}$	is the compliment of y

Integer a can be written in different representations. For example,

$$\begin{aligned}
a &= 25 && \text{decimal representation} \\
&= 0x19 && \text{hexadecimal representation} \\
&= 00011001_2 && \text{binary representation} \\
&= (0001||1001) && \text{binary block representation}
\end{aligned}$$

3.2.2 Description of ZUC

ZUC is a word-oriented stream cipher which has a 128-bit secret key $k = (k_0, k_1, \dots, k_{15})$ and a 128-bit initial vector $iv = (iv_0, iv_1, \dots, iv_{15})$ as inputs, and output is a 32-bit words key stream, where k_i, v_i are 8-bit wide, ($0 \leq i \leq 15$). *ZUC* has two stages: key initialization stage and working stage. In [7] the general structure of *ZUC* is described in Figure 3.1.

The linear feedback shift register (LFSR) has 31-bit registers $s_0, s_1, \dots, s_{15}$ which are in the range $\{1, 2, \dots, 2^{31} - 1\}$. The 31-bit registers s_i are constructed by using 8-bit secret keys k_i , 15-bit constants d_i defined in [7], and 8-bit initial vectors iv_i , i.e., $s_i = (k_i || d_i || iv_i)$, ($0 \leq i \leq 15$). Our attacks involve only the initialization stage, so we only briefly describe that stage. The working stage can be found in [7]. Denote $u = (W \oplus X_3) \ggg 1$, where W and X_3 are described in Fig 3.1. The initialization stage contains the LFSRWithInitialisationMode(u), Bitreorganization() and $F(X_0, X_1, X_2)$ described in [7] as in Algorithms 1, 3 and 2.

After constructing s_i from (k_i, iv_i, d_i) , ($0 \leq i \leq 15$), and 32-bit R_1 and R_2 are set to 0, the initialization stage works as in Algorithm 4.

begin

1.

$$v = 2^{15}s_{15} + 2^{17}s_{13} + 2^{21}s_{10} + 2^{20}s_4 + (1 + 2^8)s_0 \mod (2^{31} - 1); \quad (3.1)$$

2. If $v=0$ then set $v = 2^{31} - 1$;3. $s_{16} = v \oplus u$;4. If $s_{16} = 0$ then set $s_{16} = 2^{31} - 1$;5. $(s_1, s_2, \dots, s_{15}, s_{16}) \rightarrow (s_0, s_1, \dots, s_{14}, s_{15})$;**end****Algorithm 1:** LFSRWithInialisationMode(u).**begin**1. $W = (X_0 \oplus R_1) \boxplus R_2$;2. $W_1 = R_1 \boxplus X_1$;3. $W_2 = R_2 \oplus X_2$;4. $R_1 = S(L_1(W_{1L} || W_{2H}))$;5. $R_2 = S(L_2(W_{2L} || W_{1H}))$;where S is the S-box layer and L is the linear transformation layer in [7].**end****Algorithm 2:** F(X_0, X_1, X_2).**begin**1. $X_0 = (s_{15H} || s_{14L})$;2. $X_1 = (s_{11L} || s_{9H})$;3. $X_2 = (s_{7L} || s_{5H})$;4. $X_3 = (s_{2L} || s_{0H})$;**end****Algorithm 3:** Bitreorganization().

Fig. 3.1: General structure of ZUC [7]

Note 3.2.1.

$$s_i = (k_i || d_i || iv_i) = (k_i || 0_{23}) + (0_8 || d_i || 0_8) + (0_{23} || iv_i).$$

$$2^k s_i \mod (2^{31}-1) = s_i \lll k.$$

3.3 Weaknesses in ZUC Algorithm

In this section, we show that there exist two weaknesses in *ZUC*. Wu made an important observation as follows. Assume that:

```

begin
  for  $i:=0$  to 31 do
    1. Bitreorganization();
    2.  $u = F(X_0, X_1, X_2) \oplus X_3$ ;
    3. LFSRWithInitialisationMode( $u \gg 1$ );
  end
end

```

Algorithm 4: InitializationStage().

$k = 87, 4, 95, 13, 161, 32, 199, 61, 20, 147, 56, 84, 126, 205, 165, 148.$

$iv = 166, 166, 112, 38, 192, 214, 34, 211, 170, 25, 18, 71, 4, 135, 68, 5.$

$iv' = 116, 166, 112, 38, 192, 214, 34, 211, 170, 25, 18, 71, 4, 135, 68, 5.$

Two different initial vectors iv and iv' using the secret key k generate the same key stream output, i.e., `bfe800d5 0360a22b 6c4554c8 ...`. We say that key k has a collision if there exists such pair (iv, iv') for key k .

Lemma 3.3.1. *In ZUC, there exist keys which have a collision.*

After analyzing the construction of s_i , ($0 \leq i \leq 15$), X_0, X_1, X_2, X_3 and function LFSRWithInitialisationMode(u), we suggest to focus on Steps 3 and 4 in LFSRWithInitialisationMode(u) function which are: ($s_{16} = v \oplus u$) and (If $s_{16} = 0_{31}$, then set $s_{16} = 2^{31} - 1 = 1_{31}$). If two different iv, iv' of k generate the same u and two different v, v' such that $v \oplus u = 1_{31}$ and $v' \oplus u = 0_{31}$, then they produce the same s_{16} . Hence, the generated key streams of iv and iv' are the same. If we would like to obtain a pair (iv, iv') generating the collision, then we have only two following cases due to ZUC algorithm:

1. iv and iv' are the same at all iv_i , ($0 \leq i \leq 15$), except for iv_0 ($iv_0 \neq iv'_0$), i.e., in the initialization stage, iv and iv' generate the same $s_1, s_2, \dots, s_{16}, s_{17}, \dots$, except for s_0 .
2. iv and iv' are the same at all iv_i , ($0 \leq i \leq 15$), except for iv_1 ($iv_1 \neq iv'_1$), i.e., in the initialization stage, iv and iv' generate the same $s_0, s_2, \dots, s_{16}, s_{17}, \dots$, except for s_1 .

Since $v = 2^{15}s_{15} + 2^{17}s_{13} + 2^{21}s_{10} + 2^{20}s_4 + (1 + 2^8)s_0 \pmod{2^{31} - 1}$,
 $s_i = (k_i || d_i || iv_i) = (k_i || d_i || 0_8) + (0_{23} || iv_i)$, ($0 \leq i \leq 15$), and the only difference between these two cases are at iv_0 (or iv_1), v can be rewritten as follows:

$$\begin{aligned}
 v &= a + (1 + 2^8)((k || d || 0_8) + (0_{23} || iv_t)) \pmod{2^{31} - 1} \\
 &= a' + (1 + 2^8)(0_{23} || iv_t) \pmod{2^{31} - 1} \\
 &= a' + (0_{15} || iv_t || iv_t) \pmod{2^{31} - 1},
 \end{aligned}$$

where $a = 2^{15}s_{15} + 2^{17}s_{13} + 2^{21}s_{10} + 2^{20}s_4 \pmod{2^{31} - 1}$, $k = k_0$, $d = d_0$, $iv_t = iv_0$ in case 1 or $a = 2^{15}s_{16} + 2^{17}s_{14} + 2^{21}s_{11} + 2^{20}s_5 \pmod{2^{31} - 1}$, $k = k_1$, $d = d_1$, $iv_t = iv_1$ in case 2 and $a' = a + (1 + 2^8)(k || d || 0_8) \pmod{2^{31} - 1}$.

We prove that the structures of such v and v' must have a special property to satisfy the condition which is required for the same key stream word output.

Since

$$\begin{aligned}
 v &= a' + (0_{15} || iv_t || iv_t) \pmod{2^{31} - 1}, \\
 v' &= a' + (0_{15} || iv'_t || iv'_t) \pmod{2^{31} - 1}, \\
 v \oplus u &= 0_{31}, \\
 v' \oplus u &= 1_{31},
 \end{aligned} \tag{3.2}$$

we have

$$\begin{aligned} v - v' &= (0_{15} || iv_t || iv_t) - (0_{15} || iv'_t || iv'_t) \mod (2^{31} - 1) \\ &= (0_{15} || iv_t || iv_t) + (1_{15} || 0xFF - iv'_t || 0xFF - iv'_t) \mod (2^{31} - 1), \end{aligned}$$

and

$$v \oplus v' = 1_{31} (\Leftrightarrow v + v' = 0 \mod (2^{31} - 1)).$$

From two equations $(v - v')$ and $(v + v')$ above, we have

$$2v = (0_{15} || iv_t || iv_t) + (1_{15} || 0xFF - iv'_t || 0xFF - iv'_t) \mod (2^{31} - 1). \quad (3.3)$$

Note 3.3.2. From the equation 3.3, if the value of v and iv_t are given, then the iv'_t will be determined.

We check all the values of iv_t and iv'_t , ($iv_t \neq iv'_t$), to compute all $2v$. There are 510 different values of $2v$, which can formally be described as follows:

$$2v = \begin{cases} (1_{15} || x_1x_2x_3x_4x_5x_6x_7x_8 || x_1x_2x_3x_4x_5x_6x_7x_8), \\ (0_{15} || x_1x_2x_3x_4x_5x_6x_7x_8 || x_1x_2x_3x_4x_5x_6x_7x_8), \end{cases} \quad (3.4)$$

where $x_1x_2x_3x_4x_5x_6x_7x_8$ is the binary representation of a certain non zero 8-bit integer, and $x_i = 0$ or 1 , ($1 \leq i \leq 8$).

3.4 The IV_0 Weakness

In this section, we study the first weakness in ZUC algorithm, i.e., iv and iv' are the same at all iv_i , ($0 \leq i \leq 15$), except iv_0 . A key is called a *weak key* if it has

IV_0 weakness. We propose a divide-and-conquer algorithm (called Weak-Keys-IV0) to determine all weak keys. An optimal set (iv, iv') used for detecting a weak key with size $2^{23.5}$ will be introduced as well.

3.4.1 Weak Keys detected by Using IV_0 Weakness

All weak keys which have the collision with the difference at iv_0 are pointed out in this subsection. All weak keys are determined as follows.

The system of equations (3.2) is equal to the 2 following systems of equations with v defined in (3.4) and equation 3.1

$$\begin{aligned} v \oplus v' &= 1_{31}. & (3.5) \quad v \oplus v' &= 1_{31}. & (3.6) \\ v \oplus u &= 0_{31}. & v \oplus u &= 1_{31}. \end{aligned}$$

Once the value of v is fixed (510 values), the value of u in (9.3.2) (or (9.3.2)) is automatically derived. Since u contains all the bits of k_{15} , k_0 , iv_2 and 7 most significant bits of iv_{14} (see in Figure 3.1), all the values of those bits can be computed directly from u . Therefore, k_{15} and k_0 are known. Since the variable v in equation (3.1) contains k_0 , k_4 , k_{10} , k_{13} , k_{15} , iv_0 , iv_4 , iv_{10} , iv_{13} , iv_{15} and k_{15} , k_0 are known, we only need to consider the value k_4 , k_{10} , k_{13} , iv_0 , iv_4 , iv_{10} , iv_{13} , iv_{15} . Our aim is to compute all weak keys. To do that, we separate the key bits and the iv bits, i.e., we keep all the key bits of k_4 , k_{10} , k_{13} in the right hand side (RHS) and the iv bits of iv_4 , iv_{10} , iv_{13} , iv_{15} in the left hand side (LHS). Then we construct two sets Set_A and Set_B containing all values generated by the key bits in RHS and all values generated by iv bits in LHS, respectively. A weak key is a key which has a pair (iv, iv') generating the same key stream. If Set_C is non empty, where

$Set_C = Set_A \cap Set_B$, then there exist weak keys for checked value of v in system of equation (9.3.2) (or (9.3.2)). Based on note 3.3.2, if the value of v and iv_0 are given, then iv'_0 is automatically derived. The weak keys are those keys in RHS which contribute to Set_C . The Figure 3.2 describes the computation of weak keys.

Fig. 3.2: Computation of weak keys based on given v .

The process of determining the weak keys is as follows.

From the Figure 3.1,

$$u = (k_7^{15} k_6^{15} k_5^{15} k_4^{15} k_3^{15} k_2^{15} k_1^{15} k_0^{15} 1000111110011010 iv_7^{14} iv_6^{14} iv_5^{14} iv_4^{14} iv_3^{14} iv_2^{14} iv_1^{14})$$

$$\oplus (01101011 iv_7^2 iv_6^2 iv_5^2 iv_4^2 iv_3^2 iv_2^2 iv_1^2 iv_0^2 k_7^0 k_6^0 k_5^0 k_4^0 k_3^0 k_2^0 k_1^0 k_0^0 1000100)$$

where $k_7^{15} k_6^{15} k_5^{15} k_4^{15} k_3^{15} k_2^{15} k_1^{15} k_0^{15}$, $iv_7^{14} iv_6^{14} iv_5^{14} iv_4^{14} iv_3^{14} iv_2^{14} iv_1^{14}$, $iv_7^2 iv_6^2 iv_5^2 iv_4^2 iv_3^2 iv_2^2 iv_1^2 iv_0^2$, $k_7^0 k_6^0 k_5^0 k_4^0 k_3^0 k_2^0 k_1^0 k_0^0$ are the binary representation of k_{15} , iv_{14} , iv_2 , k_0 , respectively.

Based on the values of v in (3.4) and the systems of equation (9.3.2, 9.3.2) we have 4 following cases:

Case 1

$$v = (0000 \ 0000 \ 0000 \ 0000 \ || \ x_1x_2x_3x_4x_5x_6x_70 \ || \ x_1x_2x_3x_4x_5x_6x_7),$$

$$v' = (1111 \ 1111 \ 1111 \ 1111 \ || \ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_71 \ || \ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_7),$$

$$v \oplus u = 0_{31}.$$

From formula of u and the system equations above, we have

$$k_{15} = k_7^{15}k_6^{15}k_5^{15}k_4^{15}k_3^{15}k_2^{15}k_1^{15}k_0^{15} = 01101011,$$

$$iv_2 = iv_7^2iv_6^2iv_5^2iv_4^2iv_3^2iv_2^2iv_1^2iv_0^2 = 10001111,$$

$$k_0 = k_7^0k_6^0k_5^0k_4^0k_3^0k_2^0k_1^0k_0^0 = \bar{x}_1x_2x_3\bar{x}_4\bar{x}_5x_6\bar{x}_70,$$

$$iv_{14}(7 \dots 1) = iv_7^{14}iv_6^{14}iv_5^{14}iv_4^{14}iv_3^{14}iv_2^{14}iv_1^{14} = \bar{x}_1x_2x_3x_4\bar{x}_5x_6x_7.$$

Case 2

$$v = (0000 \ 0000 \ 0000 \ 0000 \ || \ x_1x_2x_3x_4x_5x_6x_70 \ || \ x_1x_2x_3x_4x_5x_6x_7),$$

$$v' = (1111 \ 1111 \ 1111 \ 1111 \ || \ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_71 \ || \ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_7),$$

$$v \oplus u = 1_{31}.$$

(3.7)

Hence,

$$k_{15} = 10010100,$$

$$iv_2 = 01110000,$$

$$k_0 = x_1\bar{x}_2\bar{x}_3x_4x_5\bar{x}_6x_71,$$

$$iv_{14}(7 \dots 1) = x_1\bar{x}_2\bar{x}_3\bar{x}_4x_5\bar{x}_6\bar{x}_7.$$

Case 3

$$\begin{aligned}
v &= (1000 \ 0000 \ 0000 \ 0000 \ || \ x_1x_2x_3x_4x_5x_6x_71 \ || \ x_1x_2x_3x_4x_5x_6x_7), \\
v' &= (0111 \ 1111 \ 1111 \ 1111 \ || \ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_70 \ || \ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_7), \\
v \oplus u &= 0_{31}.
\end{aligned} \tag{3.8}$$

Therefore,

$$\begin{aligned}
k_{15} &= 11101011, \\
iv_2 &= 10001111, \\
k_0 &= \bar{x}_1x_2x_3\bar{x}_4\bar{x}_5x_6\bar{x}_70, \\
iv_{14}(7 \dots 1) &= \bar{x}_1x_2x_3x_4\bar{x}_5x_6x_7.
\end{aligned}$$

Case 4

$$\begin{aligned}
v &= (1000 \ 0000 \ 0000 \ 0000 \ || \ x_1x_2x_3x_4x_5x_6x_71 \ || \ x_1x_2x_3x_4x_5x_6x_7), \\
v' &= (0111 \ 1111 \ 1111 \ 1111 \ || \ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_70 \ || \ \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6\bar{x}_7), \\
v \oplus u &= 1_{31}.
\end{aligned}$$

Hence,

$$\begin{aligned}
k_{15} &= 00010100, \\
iv_2 &= 01110000, \\
k_0 &= x_1\bar{x}_2\bar{x}_3x_4x_5\bar{x}_6x_70, \\
iv_{14}(7 \dots 1) &= x_1\bar{x}_2\bar{x}_3\bar{x}_4x_5\bar{x}_6\bar{x}_7.
\end{aligned}$$

If the value of v is given, then $k_{15}, k_0, iv_2, iv_{14}(7 \dots 1)$ are determined. Then, all determined values of k_{15}, k_0 are substituted to v in equation (3.1). All $s_i, (1 \leq i \leq 15)$, are broken into 3 parts k_i, d_i, iv_i and then we keep all unknown key bits in the RHS, move all the iv bits into LHS. The sum over all the constants in (3.1) is equal to $0x451bfe1b$, i.e., $2^{15}(0_8||d_{15}||0_8) + 2^{17}(0_8||d_{13}||0_8) + 2^{21}(0_8||d_{10}||0_8) + 2^{20}(0_8||d_4||0_8) + (1 + 2^8)(0_8||d_0||0_8) \pmod{(2^{31} - 1)} = 0x451bfe1b \pmod{(2^{31} - 1)}$.

Hence,

$$\begin{aligned} v - [2^{15}(k_{15}||0_{23}) + (1 + 2^8)(k_0||0_{23}) + 2^{15}(0_{23}||iv_{15}) + 2^{17}(0_{23}||iv_{13}) + \\ 2^{21}(0_{23}||iv_{10}) + 2^{20}(0_{23}||iv_4) + (1 + 2^8)(0_{23}||iv_0) + 0x451bfe1b] = \end{aligned} \quad (3.9)$$

$$2^{17}(k_{13}||0_{23}) + 2^{21}(k_{10}||0_{23}) + 2^{20}(k_4||0_{23}) \pmod{(2^{31} - 1)}.$$

For any k_{15} , which is a constant, in 4 cases, the key k_0 is defined by value of v . Without loss of generality, the case 1 is considered. If there exist $(k_{15}, k_0, k_{13}, k_{10}, k_4)$ in certain weak key k , then the equation (3.9) must have solution, i.e., there must exist $(k_{15}, k_0, k_{13}, k_{10}, k_4, iv_{15}, iv_{13}, iv_{10}, iv_4, iv_0)$ satisfying equation (3.9). For a given k_0 , then only one v is specified. We construct $Set_A = \{\forall(k_{13}, k_{10}, k_4): 2^{17}(k_{13}||0_{23}) + 2^{21}(k_{10}||0_{23}) + 2^{20}(k_4||0_{23}) \pmod{(2^{31} - 1)}\}$ for optimization goal. Set_A is the set of all possible values of RHS generated by all values of (k_{13}, k_{10}, k_4) . The complexity for constructing Set_A can be done in 2^{18} instead of 2^{24} if we exploit the arrangement of k_{13}, k_{10}, k_4 in Set_A . The cardinality of Set_A is less than 2^{13} . Constructing the *LWK* (List of Weak Keys) array with 2^{13} entries where each entry is defined as follows:

$$LWK[i] = \{\#(k_{13}, k_{10}, k_4) : 2^{17}(k_{13}||0_{23}) + 2^{21}(k_{10}||0_{23}) + 2^{20}(k_4||0_{23}) = i \pmod{(2^{31} - 1)}\}.$$

We have the following algorithm to filter all k_{13}, k_{10}, k_4 for each k_{15}, k_0 determined

from 4 cases above.

Input: k_{15}, k_0 , LWK array, Set_A

Output: the list of (k_{13}, k_{10}, k_4) for given (k_{15}, k_0)

begin

1. Compute Set_B which is the set of all possible values of LHS generated by all values of $(iv_{15}, iv_{13}, iv_{10}, iv_4, iv_0)$. The complexity for constructing Set_B can be done in 2^{23} instead of 2^{40} due to the same reason with Set_A . The cardinality of Set_B is less than 2^{23} ;
2. $Set_C = Set_A \cap Set_B$, the complexity of this process is 2^{13} . If Set_C is empty, then the equation (3.9) does not have any solution. If Set_C is non empty, then the list of weak keys (k_{13}, k_{10}, k_4) for a given k_{15}, k_0 is

$$= \bigcup_{j \in Set_C} LWK[j];$$

end

Algorithm 5: Weak-Keys-IV0.

After computing 4 cases, all weak keys are listed out in Table A.1 in the Appendix. There are around $2^{24.8}$ out of 2^{40} possible values of $k_0, k_4, k_{10}, k_{13}, k_{15}$ appears in Table A.1. Therefore, the total weak keys are $2^{128-40+24.8} = 2^{112.8}$ and the success probability of attack in IV_0 weakness is $2^{112.8}/2^{128} = 2^{-15.2}$.

3.4.2 The Optimal Checking Set of (iv, iv')

We introduce an optimal set of $iv_{15}, iv_{13}, iv_{10}, iv_4$ which helps us to check whether a given key is a weak key or not. Since there is no solution for the case 1 and case 4, all weak keys and the chosen (iv, iv') must satisfy the equations (3.7) or (3.8). Therefore, the $k_{15} = \{0x94, 0xeb\}$, $iv_2 = \{0x70, 0x8f\}$, and $iv_{14}(7, \dots, 1)$ is specified by k_0 (or value of v). We only need to choose the values $iv_{15}, iv_{13}, iv_{10}, iv_4$. A native approach is to try all values of $iv_{15}, iv_{13}, iv_{10}, iv_4$. We introduce an algorithm which

can help us to choose only 453 pairs (iv_{15}, iv_4) which can cover all needed values of $iv_{15}, iv_{13}, iv_{10}, iv_4$. With that optimal set (iv, iv') , we can quickly check whether a given key is weak or not.

The equations (3.9) is rewritten as follows

$$\begin{aligned} 2^{15}(0_{23}||iv_{15}) + 2^{17}(0_{23}||iv_{13}) + 2^{21}(0_{23}||iv_{10}) + 2^{20}(0_{23}||iv_4) = \\ v - [2^{15}(k_{15}||0_{23}) + (1 + 2^8)(k_0||0_{23}) + 2^{17}(k_{13}||0_{23}) + \\ 2^{21}(k_{10}||0_{23}) + 2^{20}(k_4||0_{23}) + (1 + 2^8)(0_{23}||iv_0) + 0x451bfe1b] \bmod (2^{31} - 1). \end{aligned} \quad (3.10)$$

In equation (3.10), we substitute all values of Set_A , iv_0 , and possible values of k_0 (possible values of v) to its RHS. Since the value of the LHS has the following form $(0p_1p_2 \dots p_{15}0000 \ 000000000000)$ where $p_i = \{0, 1\}, 1 \leq i \leq 15$, we filter all these received values in RHS, i.e., the possible values of RHS must appear in the same form of LHS. Hence we just keep those value which can be written in this form $(0p_1p_2 \dots p_{15}0000000000000000)$. After computing, there are 453 filtered values and all have $p_1 = 0$, i.e., we can use all the values of iv_{15} and all values of 6 most significant bits of iv_4 to cover all the filtered values. We can have a better result by using the Set_D which is described in Appendix A, which can help us cover all 453 filtered values.

For a given k_0 , then only one v is specified. If any value of iv_0 is chosen, then only one corresponding value of iv'_0 is determined by (3.3) for any v . Since there are only 100 values of k_0 , only 100 pairs (iv_0, iv'_0) are available. Hence, we need 453 pairs for $iv_{15}, iv_{13}, iv_{10}, iv_4$, 2 values for iv_2 , 2^7 values for iv_{14} and 100 pairs (iv_0, iv'_0) , i.e., we need to check $2^{23.5} (iv, iv')$. If the given key is a weak key with a certain distinguishing set of $iv_{15}, iv_{13}, iv_{10}, iv_4, iv_2, iv_{14}, iv_0$, then we can have

the values of k_{15}, k_0 and the sum of $(k_{13} + k_{10} + k_4)$, i.e., the correct k_{13}, k_{10}, k_4 is one of $262144 = 2^{18}$ combinations corresponding to k_{15}, k_0 . Since all the values of 88 remaining key bits are found by brute force, the time complexity to recover the secret key k is $2^{88+18} = 2^{106}$.

3.5 The IV1 Weakness

The second weakness in ZUC algorithm is studied in this section and we introduce two attacks based on this weakness, i.e., the AttackIV0 and the AttackIV1. The weakness is at iv_1 , where a given key has a collision if 2 different iv_1, iv'_1 are used. The most interesting finding is that by exploiting this weak point, any key is able to be recovered with the success probability 100%. The attack requires 2^{62} pairs (iv, iv') to find the collision for an arbitrary key.

The attack works as follows. For a given secret key k , the existence of a solution satisfying the equation system 3.2 implies that the key has a collision, i.e., the key can be recovered. Our aim is to prove the existence of such solution for any given key. Then, the attack will be derived after that. The equivalent equation systems to 3.2 is as follows:

$$\begin{aligned}
 v &= a' + (0_{15} || iv_1 || iv_1) \bmod(2^{31} - 1), \\
 v' &= v + (0_{15} || iv'_1 || iv'_1) - (0_{15} || iv_1 || iv_1) \bmod(2^{31} - 1), \\
 u &= v, \\
 v &= \text{is defined in 3.4,}
 \end{aligned} \tag{3.11}$$

where $a' = 2^{15}s_{16} + 2^{17}s_{14} + 2^{21}s_{11} + 2^{20}s_5 + (1 + 2^8)(k_1 || d_1 || 0_8) \bmod(2^{31} - 1)$. We shall prove that the solution for 3.11 exists using the following steps. First, we prove

that for a fixed value v and secret key k , there always exists a set of iv bits (denoted S_A) such that $v = a' + (0_{15} || iv_1 || iv_1) \bmod (2^{31} - 1)$. Then we prove that for any secret key k , there always exists a set of iv bits (called S_B) and v such that $S_A \cap S_B = \emptyset$ and $u = v$. All those results implies the solution for 3.11 exists. Note that, if v and iv_1 are known, then iv'_1 would be automatically determined by 3.3.

The flowchart of our proof is formally described as follows.

If the key k is weak at iv_1 , then for (iv, iv') we shall have the same s_{17} . From Figure3.1,

$$\begin{aligned}
 v &= (2^{15}s_{16} + 2^{17}s_{14} + 2^{21}s_{11} + 2^{20}s_5 + (1 + 2^8)s_1) \bmod(2^{31} - 1), \\
 s_{16} &= (2^{15}s_{15} + 2^{17}s_{13} + 2^{21}s_{10} + 2^{20}s_4 + (1 + 2^8)s_0) \\
 &\quad \oplus (k'_{15} || iv'_2 || k'_0 || iv'_{14}) \bmod(2^{31} - 1),
 \end{aligned}$$

where

$$\begin{aligned}
 k'_{15} &= k_{15} \oplus 01101011, \\
 iv'_2 &= iv_2 \oplus 10001111, \\
 k'_0 &= k_0 \oplus 10011010, \\
 iv'_{14} &= iv_{14}(7, \dots, 1) \oplus 1000100
 \end{aligned}
 \tag{3.12}$$

bits 0 $\rightarrow$ 4 is covered by : bits 0 $\rightarrow$ 4 of iv_0 ,
 bits 5 $\rightarrow$ 12 is covered by : 8 bits of iv_5 ,
 bits 13 $\rightarrow$ 15 is covered by : bits 5 $\rightarrow$ 7 of iv_0 ,
 bit 16 is covered by : bit 0 of iv_1 ,
 bits 17 $\rightarrow$ 24 is covered by : 8 bits of iv_{13} ,
 bits 25 $\rightarrow$ 30 is covered by : bits 1 $\rightarrow$ 6 of iv_1 .

If we run through all the values of $S_A = \{iv_0, iv_1, iv_5, iv_{13}\}$ in the LHS then all 2^{31} different 31-bit elements would be generated. Therefore there exists the value of $iv_0, iv_1, iv_5, iv_{13}$ such that the equation (3.13) holds for any secret key k and a fixed value v .

Now, we consider the equation $v \oplus u = 0$ or $u = v$, where

$$\begin{aligned}
 v &= (10000000000000000 || x_1x_2x_3x_4x_5x_6x_71 || x_1x_2x_3x_4x_5x_6x_7), \\
 u &= (((X_0 \oplus R_1) + R_2) \oplus X_3) \gg 1, \\
 X_0 &= (s_{16H} || 10101100 || iv_{15}), \\
 X_3 &= (01011110 || iv_3 || k_1 || 01001101), \\
 R_1 &= \text{is defined as in } F(X_0, X_1, X_2), \\
 R_2 &= \text{is defined as in } F(X_0, X_1, X_2).
 \end{aligned}$$

Note that the value of x_i , ($1 \leq i \leq 7$), can be 0 or 1 and $(x_1x_2x_3x_4x_5x_6x_71) \neq 0_8$.

Hence, we rewrite $v \oplus u = 0$ as follows

$$v \oplus (((X_0 \oplus R_1) + R_2) \oplus X_3) \gg 1 = 0. \quad (3.14)$$

Let the order of the bits in 31-bit number be $(30, 29, \dots, 0)$. The set $S_B = \{iv_3, iv_7, iv_{11}, (x_1x_2 \ x_3 \ x_4 \ x_5x_6x_7)\}$ above can cover all 31 bits because

bits 0 $\rightarrow$ 6 is covered by : 7 bits $x_1x_2x_3x_4x_5x_6x_7$ of v ,
 bits 7 $\rightarrow$ 14 is covered by : 8 bits of iv_{11} through $s_0()$ of R_2 ,
 bits 15 $\rightarrow$ 23 is covered by : 8 bits of iv_3 of X_3 ,
 bits 24 $\rightarrow$ 31 is covered by : 8 bits of iv_7 through $s_0()$ of R_1 .

If we use the same argument as above, then we argue that there exists $iv_3, iv_7, iv_{11}, (x_1x_2 \ x_3x_4x_5x_6x_7)$ such that $v \oplus u = 0_{31}$ and $S_A \cap S_B = \emptyset$. It means that for any secret key, there exists $v(\sim iv'_1), iv_0, iv_1, iv_3, iv_5, iv_7, iv_{11}, iv_{13}$ such that the key has a collision. For arbitrary pairs (v, iv_1) is chosen, then only one corresponding iv'_1 is determined by (3.3). The number of set (iv, iv') required for testing is 2^{62} . We construct the supporting array Set_F which is used for optimizing the algorithm which recovers the secret key k . The Set_F is constructed as follows:

1. Set_F has 2^{31} entries.
2. The entry $Set_F[i]$, $(0 \leq i \leq 2^{31} - 1)$, is the list of (k_1, k_{11}, k_{14}) such that

$$i = 2^{17}(k_{14}||0_{23}) + 2^{21}(k_{11}||0_{23}) + (1 + 2^8)(k_1||0_{23}) \bmod(2^{31} - 1).$$

The maximum number of elements (k_1, k_{11}, k_{14}) is 16 for any entry $Set_F[i]$, $0 \leq i \leq 2^{31} - 1$. Assume that we already found the (iv, iv') such that the key has collision, then we compute v from iv_1, iv'_1 by (3.3). We recover the secret key by the

algorithm that is given below:

Input: v, iv, Set_F

Output: the correct secret key $(k_0, k_1, \dots, k_{15})$

begin

forall the $k_2, k_3, k_6, k_7, k_8, k_{12}, k_0, k_4, k_{10}, k_{13}, k_{15}$ **do**

 compute s_{16} from (3.12);

forall the k_5 **do**

 compute

$$i = (v - (2^{15}s_{16} + 2^{20}s_5 + 2^{17}(0_8 || d_{14} || iv_{14}) + 2^{21}(0_8 || d_{11} || iv_{11}) + 2^{20}(0_8 || d_5 || iv_5) + (1 + 2^8)(0_8 || d_1 || iv_1))) \bmod (2^{31}-1);$$

forall the $(k_1, k_{11}, k_{14}) \in Set_F[i]$ **do**

 compute k_9 from R_1 in (3.14) ;

if the computed key is correct **then**

 Print out the correct key ;

 Stop the algorithm ;

end

end

end

end

end

Algorithm 6: AttackIV1.

The time complexity for recovering all the secret key is 2^{100} . If the key is weak

at iv_0 , then we can optimize the searching secret key better. Let us construct the set Set_G , as follows:

1. Set_G has 2^{13} entries.
2. The entry $Set_G[i]$, ($0 \leq i \leq 2^{13} - 1$), is the list of (k_{10}, k_{13}) such that $i = k_{13} + 2^4 k_{10} \bmod(2^{31} - 1)$.

The maximum number of elements (k_{10}, k_{13}) is 16 for any entry $Set_G[i]$, $0 \leq i \leq 2^{13} - 1$. Suppose that we already determine the given secret key is the weak key at iv_0 by using the optimal set. Then we can derive k_0, k_{15}, v and compute $j = v - (2^{15}s_{15} + 2^{17}(0_8 || d_{13} || iv_{13}) + 2^{21}(0_8 || d_{10} || iv_{10}) + 2^{20}(0_8 || d_4 || iv_4) + (1 + 2^8)s_0 \bmod(2^{31} - 1))$. Moreover we already found (iv, iv') with difference at iv_1 such that the key has collision and we compute the v for (iv_1, iv'_1) from (3.3). With the values of

k_0, k_{15}, j, v, iv , we can recover the secret key as follows:

Input: $v, iv, Set_F, Set_G, j, k_0, k_{15}$

Output: the correct secret key $(k_0, k_1, \dots, k_{15})$

begin

forall the $k_2, k_3, k_6, k_7, k_8, k_{12}, k_4$ **do**

 Compute $h = (j - 2^{20}(k_4 || 0_{23}) \bmod(2^{31}-1)) >> 9$;

forall the $(k_{10}, k_{13}) \in Set_G[h]$ **do**

 compute s_{16} from (3.12) ;

forall the k_5 **do**

 compute

$$i = (v - (2^{15}s_{16} + 2^{20}s_5 + 2^{17}(0_8 || d_{14} || iv_{14}) + 2^{21}(0_8 || d_{11} || iv_{11}) + 2^{20}(0_8 || d_5 || iv_5) + (1 + 2^8)(0_8 || d_1 || iv_1))) \bmod(2^{31}-1);$$

forall the $(k_1, k_{11}, k_{14}) \in Set_F[i]$ **do**

 compute k_9 from R_1 in (3.14) ;

if the computed key is correct **then**

 print out the key ;

 Stop the algorithm ;

end

end

end

end

end

end

The time complexity for recovering the secret key is 2^{70} .

3.6 Fixed Algorithm of ZUC

To fix the weak points in the ZUC, it is enough replace $v \oplus u$ by $v + u$. This modification removes the weaknesses. Since the operation $+$ is invertible, we can not obtain the collisions anymore. The current version of ZUC is the fixed algorithm of ZUC in this thesis.

3.7 Conclusion

In this chapter, we studied 2 weaknesses in ZUC stream cipher. For the first weakness, a random key can be recovered with the success probability of 2^{-16} and the time complexity of 2^{70} . Additionally, we computed an optimal set of (iv, iv') with $2^{23.5}$ pairs to quickly check the IV_0 weakness of a given arbitrary key k . For the second weakness, any random key is recovered and we need 2^{62} pairs (iv, iv') to find the collision which can launch the attack with time complexity 2^{100} . We suggest the fixed algorithm of ZUC.

4. ALGORITHM 1 OF MULTIDIMENSIONAL LINEAR CRYPTANALYSIS

In this chapter, the improvement of Algorithm 1 of Multidimensional Linear Cryptanalysis (MLC) is presented. The Algorithm 1 of MLC introduced by Hermelin *et al.* in ACISP2008 is one kind of extension of Algorithm 1 of Linear Cryptanalysis. If the Algorithm 1 of MLC is m -dimensional with N samples, then our methods reduce the time complexities from $N2^m$ to $N + 2m2^m$ for Phase 1 and 2^{2m} to $3m2^m$ for Phase 2 in the algorithm. Those methods are constructed by using the Fast Fourier Transform (FFT) and Fast Walsh Hadamard Transform (FWHT). Additionally, the upper bound of N samples is provided which implies that the MLC is a formalized algorithm of Biryukov *et al.*, which is also an extended version of Linear Cryptanalysis and is introduced in CRYPTO 2004.

This Chapter is based on the joint work with Wei Lei, Prof Wang Huaxiong and Prof Ling San. The experiments, editing this chapter was done by Wei Lei. The Method-B in this chapter was developed through the discussion with Wei Lei as well. Thank you Prof Wang Huaxiong and Prof Ling San for advising and encouraging Wei Lei and me in writing the chapter.

4.1 Evolution of Multiple Linear Cryptanalysis

Linear Cryptanalysis introduced by Matsui [65] has two algorithms, i.e., Algorithm 1 and Algorithm 2. The Algorithm 1 uses a linear approximation $g = uP \oplus vC \oplus cK$ and $Pr(g = 0) = 1/2 + \epsilon$ which covers a block cipher $E_k(\cdot)$. Here u , v , and c are the selection patterns for the plaintext P , ciphertext C and the extended key K , respectively. K is the extended key of secret key k , and ϵ is the bias of linear approximation. If a pool of sufficiently many random plaintext-ciphertext pairs (P, C) is collected, then the parity bit cK can be recovered. The number of samples needed N is proportional to d/ϵ^2 , where d is a constant that depends on the success probability.

Many papers on reducing N are published by using multiple linear approximations. In [49], m linear approximations $g_1, \dots, g_m$ with biases of $\epsilon_1, \dots, \epsilon_m$, respectively, are combined together. The linear approximations are restricted to have the same selection pattern c for the extended key, i.e., $c_1 = c_2 = \dots = c_m = c$. N can achieve an m -fold reduction while keeping the same success rate, with N proportional to $1/\sum_{i=1}^m \epsilon_i^2$. By using the same key mask in all linear approximations, it recovers at most 1-bit parity $-c \cdot K$. The paper shows that the contribution of m linear approximations reduce the number of samples needed for recovering 1 parity bit in comparison to the single approximation at the same success rate.

In 2004 Biryukov *et al.* [21] suggested a new approach to Algorithm 1. In this Algorithm 1, m statistically independent linear approximation $g_1, \dots, g_m$ are used, with biases of $\epsilon_1, \dots, \epsilon_m$ respectively and at most m parity bits are recovered with N proportional to $1/\sum_{i=1}^m \epsilon_i^2$. The paper mentioned one important observation that N would be further reduced if m' linearly dependent masks are used with $m' > m$. The observation was supported by experiments provided in [21, 29]. The advantage

is gained but there is no formal theoretical model/approach suggested except some intuitive references [29] based on linear hulls, error correcting codes, etc. In practice, m and m' are required to be not too large due to computation limits.

The Biryukov's method can significantly reduce the samples needed but the requirements on linear approximations are very hard to verify, i.e., the m approximations are required to be statistically independent. Based on the above observation, a reasonable question is whether there is any method using m linear approximations which are linearly independent in text masks but statistically correlated like in [21, 29]. Hermelin *et al.* answered this question by proposing the new framework called Multidimensional Linear Cryptanalysis (MLC) [43], i.e., the Algorithm 1 of MLC. The Algorithm 1 of MLC utilizes m linear approximations are linearly independent in text masks but they may be statistically correlated. It comprehensively explained the observation [21, 29] in terms of not only theoretical but also experimental aspects. Notably, in [43] a very important claim is written: the statistical independence assumption of [21] does not hold.

In the Algorithm 1 of MLC, the number of samples needed N is obtained based on empirical observation, i.e., the relation between N and m used linear approximations is not clear. Therefore it is not easy to estimate N even if the biases of $g_1, \dots, g_m$ are provided. In this chapter, we present the mathematical formula describing this relation which complements Algorithm 1 of MLC. This formula gives us insights on how much can be achieved with multidimensional linear cryptanalysis. We can now easily estimate N given $g_1, \dots, g_m$, hence the attack complexity. The simplicity of the formula eases cryptanalysis greatly since computing N with the original formula in [43] (for N_{key}) requires a lot of computation efforts when m is large.

Two major obstacles in Algorithm 1 of MLC are the time complexities in the distillation phase (called Phase 1) ($2^m N$) and the analysis phase (called Phase 2) (2^{2m})

of the online stage. These two obstacles restrict the number of approximations m . This limitation hinders the application of multidimensional Alg. 1 in practice when more approximations have to be used. Our second contribution is that we present a method (Method-*A*) for Phase 1 to speed up the computation for distributions to $2m2^m + mN$, using ideas from [27], and we develop Method-*B* for Phase 2 which reduces the complexity to $3m2^m$. We then arrive at an improved algorithm for the multidimensional generalization of Matsui's Alg. 1 in [43]. This new algorithm is better than previous generalizations to a number of degrees. Most importantly, the algorithm is more practical, with strong support from theoretical work in [43]. We shall show the applications to 4- and 9-round *SERPENT* as examples.

The chapter is organized as follows. Section 2 contains notations and some basic notions. Section 3 is about the statistical model and algorithm in [43]. We show briefly how the distribution of an m -dimensional vectorial Boolean function g can be constructed from m given approximations $g_1, \dots, g_m$ as Boolean functions. Section 4 contains our contributions. After giving the theoretical formula of N , we analyze the algorithm of [43] step-wise, to introduce our improvements, followed by the descriptions for each improvement. Section 5 describes the improved algorithm in details. Section 6 gives the application results on *SERPENT* and we compare them with previous cryptanalysis results [16] and [28]. In Section 7, we conclude and discuss the implications of our work.

4.2 Statistical Model and Algorithm of Hermelin et al.

We refer the reader to Chapter 2 for notations and the background on the Boolean functions, Fast Fourier Transform (FFT) and Fast Hadamard Transform (FWHT).

4.2.1 Multidimensional Generalization of Matsui's Alg. 1

We describe the core idea of the multidimensional algorithm 1 in [43]. Let there be m linear approximations $g_i := u_i P \oplus v_i C \oplus c_i K$, ($i = 1, \dots, m$). Their corresponding correlations are

$$\rho_i := 2Pr(u_i P \oplus v_i C \oplus c_i K = 0) - 1,$$

where the masks c_i for the extended key K are linearly independent. In addition, the pairs of input and output masks (u_i, v_i) are linearly independent.

Define $g := (g_1, \dots, g_m)$ with probability distribution $p = (p_0, p_1, \dots, p_{2^m-1})$, and $h := (h_1, \dots, h_m)$, where $h_i = u_i P \oplus v_i C$. We call h an experimental function, as we use two of its probability distributions, namely, the theoretical probability distribution q and the empirical probability distribution $\hat{q}$. In the attack, q is approximated by $\hat{q}$, which is computed from N samples. Let $w_i = c_i K$, $i = 1, \dots, m$ be the parity bits of the extended key K . As the c_i 's are linearly independent, $w = (w_1, \dots, w_m)$ defines a key class of K . Thus we have

$$g = h \oplus w.$$

Hence, the probability distribution q of experimental function h is a permutation of p . Since $\{c_i\}$ are linearly independent, with the 2^m possible parity vectors w , the key space K can be classified into 2^m classes. Let w^* be the correct key class, as p^{w^*} is the permutation of p corresponding to w^* , we have $q = p^{w^*}$. By [25], given w , the relationship between p^w and p is

$$p_\eta^w = \sum_{a \in V_m} (-1)^{a(\eta \oplus w)} \rho(a) = p_{\eta \oplus w}, \quad \forall \eta \in V_m. \quad (4.1)$$

The KL distance is then used to determine the correct key class w^* , by con-

structing a hypothesis testing problem of finding the closest distribution p^w with $\hat{q}$ among the 2^m possibilities of w . In [43] the following theorem is described.

Theorem 4.2.1. *Let us have an $|\mathcal{Z}|$ -ary hypothesis problem, with $|\mathcal{Z}|$ hypotheses H_w stating that the data originates from p^w , where $w \in \mathcal{Z}$ corresponds to the key. The hypothesis for which the Kullback-Leibler distance $D(\hat{q}||p^w)$ is smallest is selected. Given some success probability P_{sc} , the lower bound N for the amount of data required to give the smallest value of the statistic when the correct key is used, is given by*

$$N \approx \frac{4 \log_2 |\mathcal{Z}|}{\min_{w \neq 0} C(p^0, p^w)}. \quad (4.2)$$

Now, we shall analyze the multidimensional algorithm 1 step by step.

4.2.2 Algorithm of Hermelin et al. [43].

Note 4.2.2. In Step 2 the matrix $\mathbf{T}$ does not need to be stored. Storing a single row is sufficient. Other rows can be obtained according to Equation 4.1 by permuting this row when it's required for computation.

4.3 The Improvements

4.3.1 A Formula for N , the Number of Samples Required

Given $g = (g_1, \dots, g_m)$ and $\rho(a)$, for all $a \in V_m$, by (5), the capacity of the attack of [43] is $\min_{w \neq 0} C(p^0||p^w)$. Let $p = p^0$ and $q = p^w$, we derive the following lemma.

The proof is given in Appendix A.

Offline Stage:

Input: m linear approximations g_i , ($1 \leq i \leq m$), with correlation ρ_i and p ,
the probability distribution of g .

Output: estimated N

begin

| Compute N based on Equation 4.2 and p .

end

Online Stage:

Input: N pairs of (P, C) , where N is computed in offline stage.

Output: Correct key class

begin

1. Distillation Phase: Compute empirical probability distribution $\hat{q}$ of h using 2^m counters.

2. Analysis Phase:

Construct matrix $\mathbf{T}^{2^m \times 2^m}$, with cell $\mathbf{T}(w, \eta) = \log(\frac{\hat{q}_\eta}{p_\eta^w})$, for all $w, \eta \in V_m$.

Compute $D(\hat{q}||p^w)$, for all $w \in V_m$,

$$D = \mathbf{T}\hat{q}^T = (D(\hat{q}||p^0), \dots, D(\hat{q}||p^{2^m-1})). \quad (4.3)$$

3. Sorting Phase: Sort the list of w with $D(\hat{q}||p^w)$ in ascending order.

4. Searching Phase: Choose the correct key class as the first element in the sorted list.

end

Algorithm 8: Algorithm of Hermelin

Lemma 4.3.1.

$$C(p||q) \geq 2 \sum_{\forall a \in V_m \setminus \{0\}} \rho^2(a).$$

Combining with 4.2 we have the following theorem.

Theorem 4.3.2. *The estimation for N in the attack of [43] with m linear approximations g_i , ($i = 1, \dots, m$), where the $\{c_i\}$ are linearly independent and (u_i, v_i) are linearly independent, is given by*

$$N \approx \frac{m}{2 \sum_{\forall a \in V_m \setminus \{0\}} \epsilon^2(a)}. \quad (4.4)$$

With the new formula for N , we can now compare the multidimensional Alg. 1 with previous attacks on their data complexities, as shown in Table 4.1. The last row contains our formula for N . We can see that the framework provided by Hermelin *et al.* in [43] is able to exploit all the combined approximations systematically, as compared to Biryukov *et al.* in [21].

N	Framework	Comments
$1/\epsilon^2$	[65] Matsui	
$1/\sum_{i=1}^m \epsilon_i^2$	[49] Kaliski and Robshaw	
$1/\sum_{i=1}^m \epsilon_i^2$	[21] Biryukov <i>et al.</i>	
$m/2 \sum_{a \neq 0} \epsilon^2(a)$	[43] Hermelin <i>et al.</i>	See 4.3.2

Tab. 4.1: Comparisons of data complexities with different attack frameworks.

4.3.2 Analysis of the Multidimensional Alg. 1 in [43]

We analyze the multidimensional Alg. 1 of Hermelin *et al.* step by step and introduce our improvements.

Identifying the Bottleneck – Complexity Analysis

Offline: An estimation of N can be computed from 4.2, which is slow – to the best of our knowledge, no obvious algorithm is much faster than $3m2^m$ steps.

Online:

1 Distillation Phase: Using 2^m counters to compute $\hat{q}$ from N samples. The complexity is $2^m N$.

2 Analysis Phase: Computing D has a time complexity of $O(2^{2m})$.

If we increase m , the number of base approximations $g_1, \dots, g_m$, we may expect N to decrease [43], but the complexities in the distillation and the analysis phases suffer from exponential increase. The actual number of approximations that can be used is hence limited by the computation resources allowed. It is a trade-off.

Improving the Bottleneck

To compute the KL distance between a probability distribution $\hat{q}$ and p^w , we observe that after expanding $D(\hat{q}||p^w) = \sum_{\eta \in V_m} \hat{q}_\eta \log(\hat{q}_\eta/p_\eta^w) = \sum_{\eta \in V_m} \hat{q}_\eta \log \hat{q}_\eta - \sum_{\eta \in V_m} \hat{q}_\eta \log p_\eta^w$, the term $\sum_{\eta \in V_m} \hat{q}_\eta \log \hat{q}_\eta$ is a constant and hence does not affect the ranking of key classes. We can define $\bar{D}(\hat{q}||p^w) = \sum_{\eta \in V_m} \hat{q}_\eta \log p_\eta^w$ and use $\bar{D}(\cdot||\cdot)$ to rank the key class candidates. The list of w in the sorting phase of the online stage is now sorted by the values of $\bar{D}(\hat{q}||p^w)$ in descending order.

A new matrix $\bar{\mathbf{T}}$ is constructed as

$$\bar{\mathbf{T}}(w, \eta) = \log(p_\eta^w), \quad \forall w, \eta \in V_m. \quad (4.5)$$

In [72] matrix $\bar{\mathbf{T}}$ is proved a circulant matrix. By Theorem 2.3.3 the Fast Fourier

Transform algorithm can be applied for fast computation. For $\bar{\mathbf{T}}$, only the first column ($\bar{\mathbf{T}}(w, 0) = \log(p_0^w) = \log(p_w)$, for all $w \in V_m$) needs to be stored. The memory requirement is 2^m . $\bar{\mathbf{T}}$ is used for computing $\bar{D} = \bar{\mathbf{T}}\hat{q}^T$.

Our Improvements:

Offline: We present a formula for N based on p , without using (5).

Online:

Step 1: We present Method-A to calculate $\hat{q}$ from N samples given. The complexity is $2m2^m + mN$.

Step 2: We present Method-B to compute $\bar{D}$. The complexity is $3m2^m$.

4.3.3 Fast Computation of Empirical Distribution – Method A

Method A is for computing the empirical distribution $\hat{q}$ of the experimental function $h = (h_1, \dots, h_m)$ from N samples (P, C) . The proof of Method A is described as follows:

Method A:

- Step 1: Construct the vector E from N samples (P, C) which is used for computing the correlations of the combined linear approximations $\hat{\gamma}$ with time complexity of mN .
- Step 2: Compute $\hat{\gamma} = SE^T$, then construct the vector R with time complexity of $m2^m$ by using the FWHT.
- Step 3: Compute $\hat{q} = SR^T$ with time complexity of $m2^m$ by using the FWHT.

First, the correlations of the combined linear approximations bh , for all $b \in V_m$,

$$\hat{\gamma}(b) = Pr(bh(P, C) = 0) - Pr(bh(P, C) = 1),$$

are calculated. Then the probability distribution $\hat{q}$ is computed from $\hat{\gamma}(b)$ by Equation 2.9 .

We show how to compute $\hat{\gamma}(b)$, for all $b = (b_1, \dots, b_m) \in V_m$ from N samples:

$$\begin{aligned}\hat{\gamma}(b) &= \hat{\gamma}\left(\bigoplus_{i=1}^m b_i h_i\right) = \frac{\sum_{j=1}^N (-1)^{\bigoplus_{i=1}^m b_i h_i(P_j, C_j)}}{N} \\ &= \sum_{a \in V_m} (-1)^{\bigoplus_{i=1}^m b_i a_i} \frac{T_a}{N} = \sum_{a \in V_m} (-1)^{ba} \frac{T_a}{N},\end{aligned}$$

where $a = (a_1, \dots, a_m)$ and $T_a = \#\{(P_j, C_j), j = 1, \dots, N : h_i(P_j, C_j) = a_i\}$.

Let $S^{2^m \times 2^m}$ be the matrix defined by $S(b, a) = (-1)^{ba}$, and let $E = (\frac{T_0}{N}, \dots, \frac{T_{2^m-1}}{N})$, and $\hat{\gamma} = (\hat{\gamma}_0, \dots, \hat{\gamma}_{2^m-1})$, then

$$\hat{\gamma}(b) = \sum_{a \in V_m} S(b, a) E_a, \forall b \in V_m, \text{ or } \hat{\gamma} = S E^T.$$

Since $\mathbf{S}$ is a Hadamard matrix, we can apply the Fast Walsh-Hadamard Transform algorithm for computing $\hat{\gamma}(b)$ for all $b \in V_m$ with complexity $m2^m$, and the storage is $\mathcal{O}(2^m)$. The complexity for computing the counter vector $T = (T_1, \dots, T_{2^m})$ is mN , by evaluating the m base approximations against each of the N samples.

Construct the vector $R = (R_0, \dots, R_{2^m-1})$ with $R_b = 2^{-m} \hat{\gamma}(b)$, for all $b \in V_m$. From Equation 2.9 we have

$$\hat{q}_\eta = \sum_{b \in V_m} (-1)^{\eta b} R_b, \forall \eta \in V_m, \text{ or } \hat{q} = S R^T.$$

The Fast Hadamard Transform is used again to compute $\hat{q}$. Hence, the total complexity is $mN + 2m2^m$ for computing $\hat{q}$ from N samples.

4.3.4 Fast Computation of Kullback-Leibler Distance – Method B

Method-B is used to compute the vector $\bar{D}$ in 4.3 with modified matrix $\bar{T}$ in 4.5 based on the idea of circulant matrix in [29]. The proof of the following theorem can be found in [72].

Theorem 4.3.3. *The matrix $\bar{T}$ is level- m circulant with type $\underbrace{(2, 2, \dots, 2)}_{m\text{-times}}$.*

Method-B

: From Theorems 2.3.3, 2.3.4, and 4.3.3 we have

$$\bar{\mathbf{T}} = \mathbf{F}^* \text{diag}(\lambda) \mathbf{F}, (\mathbf{F}^* \mathbf{F} = \mathbf{I}), \quad (4.6)$$

$$\lambda = \mathbf{F} \bar{\mathbf{T}}(:, 1) \sqrt{2^m}, \quad (4.7)$$

and

$$\bar{D} = (\bar{D}(\hat{q}||p^0), \dots, \bar{D}(\hat{q}||p^{2^m-1})) = \bar{\mathbf{T}} \hat{q} = (\mathbf{F}^* (\text{diag}(\lambda) (\mathbf{F} \hat{q}))).$$

Applying the Fast Fourier Transform three times for $\mathbf{F}$, $\text{diag}(\lambda)$, and $\mathbf{F}^*$ we get all the values $\bar{D}(\hat{q}||p^w)$, for all $w \in V_m$. The complexity is $3m2^m$.

4.4 Efficiency of the Improved Algorithm

The improved algorithm is described in the Algorithm 9:

The improved results are presented in Table 4.2 to compare with previous results on the complexities of the distillation phase and the analysis phase, following [43]. In practise mN is always larger than $2m2^m$, and we estimate $\mathcal{O}(2m2^m + mN) = \mathcal{O}(mN)$.

Offline Stage:

Input: m linear approximations $g_1, \dots, g_m$ and probability distribution p and $\rho(a)$ of g , for all $a \in V_m$

Output: estimated N and computed λ of matrix $\bar{\mathbf{T}}$

begin

1. Compute N by 4.3.2.
2. Compute the eigenvalue vector λ of matrix $\bar{\mathbf{T}}$ based on 4.7 and p .

end

Online Stage:

Input: N samples (P, C) .

Output: The correct key class.

begin

1. Distillation: Compute empirical probability distribution $\hat{q}$ of h from N by Method-A.
2. Analysis: From λ and $\hat{q}$, compute vector $\bar{D}$ by Method-B.
3. Sorting: Sort the key classes w with $\bar{D}(\hat{q}||p^w)$ in descending order.
4. Searching: Choose the correct key class as the first element in the sorted list.

end

Algorithm 9: Improved Algorithm 1 of MLC.

Following the definitions of $N_{s.i.}$ and N_{plain} in [43], $N < N_{s.i.} < N_{plain}$, $m < m'$ (also see Table 4.1), we have significant improvement in the time complexities for the distillation phase and the analysis phase. In addition, we have proved that the multidimensional algorithm 1 requires fewer samples than Biryukov *et al.*[21]. The same result was observed in [43] with only empirical evidence.

	Distillation Phase				Analysis Phase			
	Plain	Biryukov	Hermelin	Method-A	Plain	Biryukov	Hermelin	Method-B
Data	$\mathcal{O}(N_{plain})$	$\mathcal{O}(N_{s.i.})$	$\mathcal{O}(N)$	$\mathcal{O}(N)$	-	-	-	-
Time	$\mathcal{O}(mN_{plain})$	$\mathcal{O}(m'N_{s.i.})$	$\mathcal{O}(2^m N)$	$\mathcal{O}(mN)$	$\mathcal{O}(m2^m)$	$\mathcal{O}(m'2^m)$	$\mathcal{O}(2^{2m})$	$\mathcal{O}(m2^m)$
Mem	$\mathcal{O}(m)$	$\mathcal{O}(m')$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$	$\mathcal{O}(2^m)$

Tab. 4.2: Complexity comparison between different algorithms

4.5 Application to Cryptanalysis

The reader is referred to [72] for the advantage of improved algorithm in 4- and 9-round *SERPENT* cryptanalysis. All the results received are the best according to our knowledge on 4- and 9-round *SERPENT* by using the improved algorithm.

4.6 Conclusions

In this chapter, we have presented a new formula for N , in terms of $\rho(a)$ and m , for the $|\mathcal{Z}|$ -ary hypothesis testing problem in the multidimensional generalization of Matsui's Algorithm 1 in [43]. The number of known plaintext needed can now be computed from $\rho(a)$ directly, whereas it is harder to compute $\min_{w \neq 0} C(p^0, p^w)$ in the original formula.

Method-A and Method-B have been presented to compute the empirical distribution and Kullback-Leibler distance, as improvements to the multidimensional Alg. 1 in [43]. A significant reduction of time complexity in the distillation and

analysis phases can be achieved. Breaking these bottlenecks allows many more base approximations to be used.

As shown in the case of 4- and 9-round *SERPENT* [72], the increase in m brings significant reduction on the data complexity and the time complexity. We expect this improved algorithm to outperform previous multiple linear cryptanalysis. We observed that the framework of Hermelin *et al.* [43], with our improvement, solves the problem of Biryukov *et al.* [21] that the statistical independence assumption cannot in general be guaranteed in practice, and in fact, does not need to be guaranteed, because there exists a large number of combined approximations with non-negligible correlations. Meanwhile, the series of experiments in [28] and [29] provide excellent examples for this argument.

It also implies that, for block cipher designs, bounding the maximum correlation for any single linear characteristic is not sufficient to claim security. Especially for SPN block ciphers with small S-boxes, a single linear trail with multiple active S-boxes in the first or last round can be modified to have many approximations, exponential in the number of active S-boxes of outer rounds. When these approximations are with similar magnitude of correlations, multidimensional linear cryptanalysis as a systematic way to exploit these combined correlations can reduce the attack complexity greatly. It's worthy for designers to have larger security margins or to try to develop specific mechanisms to prevent an attacker from forming an exponential number of valid linear approximations.

5. ALGORITHM 2 OF MULTIDIMENSIONAL LINEAR CRYPTANALYSIS

In this chapter, we study the improvement of time complexity in Algorithm 2 of MLC based on FFT and FWHT. If the algorithm is m -dimensional and reveals l bits of the last round key with N plaintext-ciphertext pairs, then the time complexity is reduced from $\mathcal{O}(mN2^l)$ to $\mathcal{O}(N + \lambda 2^{m+l})$, where λ is $3(m + l)$ or $4m + 3l$. The resulting attacks are the best known key recovery attacks on 11-round and 12-round *SERPENT*. The data, time, and memory complexities of the previously best known attack on 12-round *SERPENT* are reduced by factor of $2^{7.5}$, $2^{11.7}$, and $2^{7.5}$, respectively. This chapter also provides some experiments of the improved Algorithm 2 in multidimensional linear cryptanalysis on 5-round *SERPENT*. The editing of this chapter was based on the comments of Prof Wu Hongjun and Prof Wang Huaxiong.

5.1 Algorithm 2 of Linear Cryptanalysis

In 1993, Matsui [65] introduced the linear cryptanalysis and two algorithms, Algorithm 1 and Algorithm 2. These two algorithms exploit block cipher's linear approximation between the plaintext P , the ciphertext C and the secret key with a certain probability. Algorithm 2 is the modified version of Algorithm 1 by relaxing the first round and/or the last round of the linear approximation of Algorithm 1.

Algorithm 2 can recover multiple secret key bits instead of one bit in Algorithm 1 with the same number of samples N required by the attack. However, the time complexity of the distillation phase of Algorithm 2 is much higher than that of Algorithm 1. Without loss of generality, the substitution permutation network (SPN) n -round block cipher is considered, the last round of cipher is relaxed with l -bit subkey K_n of the last round key involved in the attack and N samples given. The time complexity for recovering l -bit K_n is $\mathcal{O}(2^l N)$. Matsui[66] suggested a method to reduce the time complexity to $\mathcal{O}(N + 2^{2l})$ by using one pre-computed table T . In 2007, Collard [27] *et al.* realized that table T is a *circulant* matrix and applied the FFT to this table to reduce the time complexity to $\mathcal{O}(N + 3l2^l)$.

There are several important papers [21, 43, 49] which extended Matsui's algorithms, i.e., m linear approximations are used instead of one linear approximation in the attack. By combining m linear approximations together in the attack, the number of samples N is expected to be reduced. Two of the most outstanding models are the full Biryukov's model [45] and Hermelin's model [44] which is known as multidimensional linear cryptanalysis. The time complexity of the extended Algorithm 2 of these models is $\mathcal{O}(mN2^l)$. Even if we apply the algorithm in [27] to the full Biryukov's model, the time complexity of the extended Algorithm 2 is $\mathcal{O}(2^m N + 3l2^{l+m})$, which is still too high. Due to the high time complexity, the extended Algorithm 2 of these models can not be used to attack as many rounds as Algorithm 2 of Matsui or the extended Algorithm 1 does.

The extended Algorithm 2 in [44] computes the information for the attack by using a very natural algorithm which has a high computation cost, i.e., $\mathcal{O}(mN2^l)$. Our main contribution is to show that by applying the FFT and FWHT to the extended Algorithm 2, the time complexity of the attack is reduced to $\mathcal{O}(N + \lambda 2^{m+l})$, where λ is $3(m + l)$ or $4m + 3l$. Firstly, we work with N pairs of samples

(P, C) to extract the information for the attack only once. Secondly, we use two assistant tables to compute the information extracted from N samples. Finally, we apply FFT, FWHT to these tables to derive the correct key from 2^l guessed keys. Based on the steps above, we introduce two methods, *method 1* and *method 2*, for two cases of the extended Algorithm 2 to reduce the time complexity to $\mathcal{O}(N + \lambda 2^{m+l})$, where λ is $3(m + l)$ or $4m + 3l$. Applying these methods to the extended Algorithm 2, we attack 11-round *SERPENT* with 2^{116} data complexity, $2^{107.5}$ time complexity, 2^{104} memory complexity. The previously best known result on 11-round *SERPENT* is given in [27] with 2^{118} data complexity, $2^{114.3}$ time complexity, 2^{108} memory complexity. Therefore, the attack in this chapter is the best on 11 rounds. For 12-round *SERPENT*, we have two attacks. The first reduces the data and time complexity in [33] by factors of $2^{5.5}$ and $2^{19.4}$, respectively, but the memory requirements are higher by a large factor of 2^{100} . In comparison with [33], the second attack reduces the data, time, and memory complexity by $2^{7.5}$, $2^{11.7}$, and $2^{7.5}$, respectively. Hence our second attack on 12 rounds is much better than [33].

This chapter is organized as follows. Section 2 introduces the background as well as some notations and definitions. Section 3 gives the brief description of Algorithm 2 of the full Biryukov model and of the Hermelin model. In this section, the algorithm with time complexity $\mathcal{O}(mN2^l)$, which is used to compute the information for determining the correct key, is also explained. Then, Section 4 describes two efficient computing methods for two cases of the extended Algorithm 2. The experimental results on 5-round *SERPENT* and the best cryptanalytic results against 11-round and 12-round *SERPENT* are reported in Section 5. Lastly, Section 6 concludes this chapter

5.2 Algorithm 2 of Multidimensional Linear Cryptanalysis

The m -dimensional linear cryptanalysis based on m linear approximations is introduced by Hermelin *et.al* [43]. In the attack, 2^m linear approximations which are the combinations of m linear approximation are exploited to reduce the data complexity. We refer the reader to chapter 2 and 4 for review of MLC and another background, which are needed for our efficient computing methods. Here we present the extended Algorithm 2 of MLC of the full Biryukov's model and the Hermelin's model and we explain the high time complexity of the extended Algorithm 2 afterwards.

Analysis on Algorithm 2 of the full Biryukov's Model and Hermelin's Model:

Algorithm 2 of MLC is the extension of the Matsui's Algorithm 2 [65] by using 2^m linear approximations constructed from m base linear approximations. Let us consider the SPN block cipher with n rounds. Let $u \cdot P \oplus v \cdot C_{n-1} \oplus w \cdot \bar{K}$ be the $(n-1)$ -round linear approximation of the block cipher, where u , v , w are the selection patterns of the plaintext P , the output C_{n-1} of $(n-1)$ -th round, and the inner key $\bar{K}$, respectively. In the Matsui's Algorithm 2, this linear approximation is extended to n rounds by replacing $v \cdot C_{n-1} = z \cdot S^{-1}(K_n \oplus C_l)$, where z is a selection pattern corresponding to v on the input of the last round, S^{-1} is the inverse of the last S-box layer, K_n is the l bits out of the last round key and C_l is the l bits out of the ciphertext C involved in the attack. Denote $z \cdot S^{-1}(K_n \oplus C_l)$ by $f(K_n \oplus C_l)$, then we have the following linear approximation which is used in Matsui's Algorithm 2:

$$u \cdot P \oplus f(K_n \oplus C_l) \oplus w \cdot \bar{K}.$$

Since the above linear approximation has correlation, the linear approximation $u \cdot P \oplus f(K_n \oplus C_l)$ also has correlation and then Matsui's Algorithm 2 [65] will determine the right key K_n based on this linear approximation. Instead of using one linear

approximation, the extended Algorithm 2 uses many linear approximations. Let us assume that the extended Algorithm 2 is m -dimensional and recovers l -bit subkey K_n of the last round key. The m linear approximations used in the extended Algorithm 2 as base linear approximations are $g_i := u_i \cdot P \oplus f(K_n \oplus C_l)$ for $i = 1, \dots, m$, where u_i are the selection patterns of plaintext. Define the vectorial boolean function $g = (g_1, \dots, g_m)$. Then, $a \cdot g := a \cdot (u_1 \cdot P, \dots, u_m \cdot P) \oplus f(K_n \oplus C_l)$ are combined approximations with the correlation $\rho(a)$ for $a \in V_m$.

The aim of the extend Algorithm 2 is to exploit 2^m combined approximations $a \cdot g$ in order to reduce the the data complexity N , i.e., [44] $N \sim \mathcal{O}(1/c)$, where $c = \sum_{\forall a \in V_m, a \neq 0} \rho^2(a)$ is called a capacity of the system. In practice, the correlation $\rho(a)$ of $a \cdot g$ is computed by *Piling-Up Lemma* [43]. The probability distribution $p = (p_0, \dots, p_{2^m-1})$ of g is not uniformly distributed. The distillation phase [21, 25] working on N samples (P, C) helps us to derive the correct K_n from 2^l guessed keys K . The time complexity of this phase is the major factor in the total time complexity of the attack. Let $\mathbf{p}^{2^l \times 2^m}$ be the matrix in which each row is empirical probability distributions of g for each candidate key K , i.e., $\mathbf{p}[K, \cdot] = (\mathbf{p}[K, 0], \dots, \mathbf{p}[K, 2^{m-1}])$ ($\forall K \in V_l$). In [45], Hermelin *et al.* proved that the full Biryukov's model is equal to the convolution model and the distillation phase of the convolution model uses the same algorithm with the Hermelin's model of Algorithm 2 to compute the empirical m -dimensional probability distribution of g , i.e., $\mathbf{p}$. The algorithm in [45] is described in Algorithm 10:

In Hermelin's algorithm, we use only m linear approximations $g_1, \dots, g_m$ and the distillation phase computes all values of $\mathbf{p}[K, \cdot]$ ($\forall K \in V_l$) with $mN2^l$ time complexity. Based on the *Wrong Key Hypothesis* [44], if K is the wrong key, its $\mathbf{p}[K, \cdot]$ is uniformly distributed and the right key K 's $\mathbf{p}[K, \cdot]$ is non uniformly distributed. The χ^2 or *LLR* statistic [44] is applied to determine what kind of probability dis-

Input: N pairs of plaintext-ciphertexts $\{(P_1, C_1), \dots, (P_N, C_N)\}$,

$g = (g_1, \dots, g_m)$.

Output: the matrix $\mathbf{p}$ of empirical probability distribution for all candidate $K_n \in V_l$.

begin

for $K \in V_l, \eta \in V_m$ **do**

 | set $\mathbf{p}[K, \eta] = 0$.

end

for $K \in V_l, t := 0, \dots, N$ **do**

 | $\eta = g(P_t, C_t, K) = (g_1(P_t, C_t, K), \dots, g_m(P_t, C_t, K))$. ;

 | $\mathbf{p}[K, \eta]++$. ;

end

for $K \in V_l, \eta \in V_m$ **do**

 | $\mathbf{p}[K, \eta] = \mathbf{p}[K, \eta]/N$.

end

end

Algorithm 10: The Hermelin's algorithm to compute the $\mathbf{p}$ with given N samples plaintext-ciphertext.

tribution for all $\mathbf{p}[K, \cdot]$ ($\forall K \in V_l$). Then, the correct key K is determined by the information received from the chosen statistic.

Although the algorithm used to compute all $\mathbf{p}[K, \cdot]$ ($\forall K \in V_l$) directly from N samples (P, C) is very elegant and can be easily implemented, its complexity is much higher in comparison to the extended Algorithm 1 [45, 72] or to Algorithm 2 of Matsui. In cryptanalysis, it is the bottleneck in terms of time complexity. Therefore the extended Algorithm 2 is not comparable to Algorithm 2 of Matsui in terms of the number of rounds attacked or time complexity.

5.3 Efficient Computation of Distillation Phase of extended

Algorithm 2 of Matsui

In this section, we describe two efficient computing methods, *method 1* and *method 2*, for two cases in multidimensional linear cryptanalysis. These methods use Lemma

2.3.5, Corollary 2.3.6, the *FFT*, *FWHT* and two pre-computed tables T, E to avoid repeatedly working on N samples and 2^l guessed keys K_n in order to reduce the time complexity from $\mathcal{O}(mN2^l)$ to $\mathcal{O}(N + \lambda 2^{m+l})$, where λ is a positive number which is specified for each case.

5.3.1 Case 1 and Method 1

In the first case, we have N pairs of samples (P, C) and m base linear approximations $g_i := u_i \cdot P \oplus f(K_n \oplus C_l)$ for $i = 1, \dots, m$, where C_l is l bits out of ciphertext C involved in attack. All m base approximation g_i share one function $f(\cdot)$ and m selection patterns of plaintext $u_1, \dots, u_m$ are linearly independent. Suppose $g = (g_1, \dots, g_m)$, we then need to compute the empirical probability distribution matrix $\mathbf{p}$ of g .

Lemma 5.3.1. *For the first case, there exists a method (method 1) which computes the empirical probability distribution $\mathbf{p}$ of g with $\mathcal{O}(N + (4m + 3l)2^{m+l})$ time complexity.*

The proof and description of *method 1* are presented in Appendix C.

5.3.2 Case 2 and Method 2

In this case, the first and the last round keys of cipher are considered in the attack. We have N pairs of samples (P, C) and m base linear approximations $g_i := f_1^i(K_1 \oplus P_{l_1}) \oplus f_2(K_n \oplus C_{l_2})$ for $i = 1, \dots, m$, where K_1 is l_1 bits of the first round key, P_{l_1} is l_1 bits of the plaintext P , K_n is l_2 bits of the last round key, C_{l_2} is l_2 bits of the ciphertext C , f_1^i, f_2 ($i = 1, \dots, m$) are the boolean functions in [27, 65]. The functions $f_1^i(\cdot)$ are constructed in the same way of $f_2(\cdot)$ for $i = 1, \dots, m$. Suppose $g = (g_1, \dots, g_m)$, we then need to compute the empirical probability distribution matrix $\mathbf{p}$ of g .

Lemma 5.3.2. *For the second case, there exists a method (method 2) which computes the empirical probability distribution $\mathbf{p}$ of g with $\mathcal{O}(N + 3(l_1 + l_2 + m)2^{l_1+l_2+m})$ time complexity.*

The proof and description of *method 2* are presented in Appendix C.

5.4 Results on Cryptanalysis of SERPENT

In this section, the experimental results of 5-round *SERPENT* are reported first. *SERPENT* [16] is the SPN block cipher which is one of the five AES finalists. *SERPENT* has 128-bit block size and supports 128-bit, 192-bit and 256-bit keys. Based on experimental results, we manage to confirm some facts claimed in [27] for the extended Algorithm 2. Then, we describe our attacks against 11-round and 12-round *SERPENT*. The cryptanalytic results show that the multidimensional linear cryptanalysis can reduce the number of samples N and the time complexity in comparison to other methods, i.e., the linear cryptanalysis of Matsui and the differential-linear cryptanalysis on *SERPENT* [33].

5.4.1 Experimental Results on 5-round SERPENT

In this subsection, the experiment on 5-round *SERPENT* aims to confirm the results in [25, 44] by using the extended Algorithm 2 with *method 1*. We call the extended Algorithm 2 with *method 1* or *method 2* improved extended Algorithm 2. For simplicity, the χ^2 statistic is applied to 4-, 7-, 10-dimensional linear cryptanalysis with $N = 2^{20}, 2^{21}, \dots, 2^{26}$ and $l = 12$. The capacities are $2^{-23}, 2^{-22}, 2^{-21}$ for 4-, 7-, 10-dimensional linear cryptanalysis, respectively. If the correct l -bit key K_n has rank r out of 2^l possible guessed keys, then the attack obtains an $(l - \log_2 r)$ -bit advantage over exhaustive search [25]. We test fifty keys and compute the advantage [21, 25] for

$\log_2(N)$	20	21	22	23	24	25	26
m=4	1.30	1.45	1.60	1.64	1.65	1.65	1.55
m=7	1.46	1.50	1.39	1.19	1.29	1.35	1.37
m=10	1.21	1.19	1.30	1.25	1.38	1.55	1.52

Tab. 5.1: The average of advantages on 5-round *SERPENT*.

each key. Then, we construct the comparison table of average advantages for 4-, 7-, and 10-dimensional linear cryptanalysis for each set of N pairs. Table 5.1 displays the average of advantages for each set of N samples and 4-, 7-, 10-dimensional linear cryptanalysis.

The results confirm the facts that the extended Algorithm 2 with χ^2 statistic does not work as well as expected and when $m = 4$ the average of advantages is the highest. Furthermore, the experiments tell us that, given capacity c and if $N = 2^2/c$, then the advantage ≥ 1 of certain key has more than 50% success probability. According to the experimental results in [25], if the combination of *LLR* statistic and maximum *KL* is used instead of χ^2 statistic, then the multidimensional linear cryptanalysis works better, i.e., the average of advantages is much higher for each set of N samples.

5.4.2 Cryptanalysis of 11-round and 12-round reduced *SERPENT*

Before going to the cryptanalysis of 11-round and 12-round *SERPENT*, we recall the fact that both methods do not require any decryption or encryption operation as that in Algorithm 2 of Matsui or in the differential-linear cryptanalysis [33]. We only need to extract the subset of bits involved in the plaintext P and the ciphertext C to compute the table E . The linear cryptanalysis can reach 11-round reduced *SERPENT* by Algorithm 2 of Matsui based on the 9-round linear approximation [16, 27, 28]. The improved extended Algorithm 2 uses the 9-round linear approx-

Round		complexity		
		data	time	memory
11	Lin.cryptanalysis [28]	2^{118} KP	2^{166} En	2^{121}
	Lin.cryptanalysis [28]	2^{118} KP	$2^{173.5}$ En	2^{97}
	Lin.cryptanalysis [27]	2^{118} KP	$2^{114.3}$ En	2^{108}
	MultiDim.cryptanalysis [this chapter](method 1)	2^{116} KP	$2^{107.5}$ En	2^{104}
	MultiDim.cryptanalysis [this chapter](method 1)	2^{118} KP	$2^{109.5}$ En	2^{100}
12	Differential-Lin.cryptanalysis [33]	$2^{123.5}$ CP	$2^{249.4}$ En	$2^{128.5}$
	MultiDim.cryptanalysis [this chapter](method 2)	2^{118} KP	$2^{228.8}$ En	2^{228}
	MultiDim.cryptanalysis [this chapter](method 1)	2^{116} KP	$2^{237.5}$ En	2^{121}

En - Encryptions, KP - Known Plaintexts, CP - Chosen Plaintexts.

Tab. 5.2: Comparison of the attacks against reduced-round *SERPENT*.

imation in [26, 27] which is called *9-round linear approximation of Collard*. The 10-round linear approximation is constructed by adding one more round before the above 9-round linear approximation. The 11-round linear approximation, which is constructed from this 10-round linear approximation with the last round S_4 relaxed, is as follows:

$$\underbrace{S_2 \longrightarrow \underbrace{S_3 \longrightarrow \dots \longrightarrow S_3}_{\text{9-round linear approximation of Collard}} \longrightarrow \underbrace{S_4}_{\text{round relaxed}}}_{\text{10-round}} .$$

In S_2 at the first round, there are 15 active S-boxes. The output masks of these active S-boxes described in hexadecimal number are:

S-box	0	3	5	6	9	10	11	13	15	16	17	24	28	29	30
output-mask	e	d	7	c	3	6	4	b	4	b	2	1	1	e	8

Based on the observation on active S-boxes of S_2 , we choose 56 linearly independent selection patterns of plaintext $u_1, \dots, u_{56}$, i.e., we have 56-dimensional linear cryptanalysis attack.

If S_4 at the 11-th round has 12 active S-boxes, then in the 11-th round, the number of key bits l is 48 and the capacity c of 56-dimensional system is computed as follows:

$$\begin{aligned}
 c &= \sum_{a \neq 0, a \in V_{56}} \rho^2(a) \\
 &= (2^{-57})^2 \sum_{m=0}^{11} \mathcal{C}_m^{11} 8^m 2^{11-m} 4^4 (2^{-4m} 2^{-2(11-m)} 2^{-8}) \\
 &= 2^{-114}.
 \end{aligned} \tag{5.1}$$

Let the number of samples required $N = 2^{116}$. If *method 1* is used, then the total time complexity is $(N/352 + 2^{111.6}/352)$ which is equal to $2^{107.5}$ 11-round *SERPENT* encryptions and the memory will be 2^{104} .

If S_4 at the 11-round has 11 active S-boxes, then in the 11-th round, l is 44 and the capacity is 2^{-116} . Let N be 2^{118} , then the total time complexity is $(N/352 + 2^{107.5}/352)$ which is equal to $2^{109.5}$ 11-round *SERPENT* encryptions and the memory is 2^{100} .

We attack the 12-round *SERPENT* by using the above 11-round linear approximation and relaxing 12-th round S_4 , which has 11 active S-boxes.

$$\underbrace{S_1 \longrightarrow S_2 \longrightarrow \underbrace{S_3 \longrightarrow \dots \longrightarrow S_3}_{\text{9-round linear approximation of Collard}} \longrightarrow \underbrace{S_4}_{\text{round relaxed}}}_{\text{11-round}}.$$

The capacity is 2^{-116} . Let $N = 2^{118}$, $l_1 = 128$ bits and $l_2 = 44$ bits. Based on *method 2*, the total time complexity is $(N/384 + 2^{237.4}/384)$ which is equal to $2^{228.8}$ 12-round *SERPENT* encryptions and the memory is 2^{228} .

We can attack 12-round *SERPENT* by *method 1* by relaxing the first and the 12-th rounds. We search for all 2^{128} possible keys in S_1 at the first round and use *method 1* for the other 11 rounds similarly to the case $N = 2^{116}$. Then the time and memory complexity are $2^{237.5}, 2^{121}$. Table 5.2 shows that the improved extended Algorithm 2 is better than all the previously known algorithms, i.e., linear cryptanalysis and differential-linear cryptanalysis, on 11-round and 12-round *SERPENT*.

5.5 Conclusion

We studied two methods to reduce the time complexity in distillation phase of extended Algorithm 2 from $\mathcal{O}(mN2^l)$ to $\mathcal{O}(N + \lambda 2^{m+l})$, where m is the number of dimension of the attack, N is the number of samples needed, l is the number of key bits in the first and/or in the last rounds. These methods are introduced when we combine the Corollary 2.3.6, the Lemma 2.3.5, exploiting the repeated structure of data and key and using 2 assistant pre-computed tables T, E . Applying FFT, FWHT to the tables T, E , we have two efficient computing methods to determine the correct key out of 2^l guessed keys.

We have simulated the experiments on 5-round *SERPENT* to check the improved extended Algorithm 2 and to confirm claims in [25, 44]. Based on the results of the experiments, we develop attack on 11-round, 12-round *SERPENT* by using the improved extended Algorithm 2. These results are the best among those reported so far. On 12-round *SERPENT*, we can reduce the data complexity, time and memory complexity of the previously best known attack by factors of $2^{7.5}, 2^{11.7}$ and $2^{7.5}$, respectively. The result of 11-round *SERPENT* is even better than the best currently reported, i.e., data complexity, time complexity, and memory complexity are reduced by factor of $2^2, 2^7$, and 2^4 , respectively. The improved extended

Algorithm 2 is competitive to the extended Algorithm 1 and Algorithm 2 of Matsui in terms of data complexity, time complexity and the number of rounds attacked. The extended Algorithm 2 usually involves many active S-boxes in the outer round(s) because it uses many linear approximations. Intuitively, it is easy to reach the limit of complexity in terms of time complexity and memory complexity. It implies that the bound of number of rounds attacked in improved extended Algorithm 2 is close to that in Algorithm 2 of Matsui somehow.

6. RELATED KEY ATTACK ON KASUMI CASE A5/3

This chapter is the joint work with Dr Matt Robshaw and Prof Wang Huaxiong. I appreciate Dr Matt Robshaw who introduced me the problem, i.e., the security proof KASUMI cipher with 64-bit key in context of Related Key Attack. Thank you Dr Matt Robshaw and Prof Wang Huaxiong for editing this chapter.

6.1 Introduction

Underpinning the success of mobile phone telephony are a range of authentication and confidentiality services. For confidentiality, several encryption algorithms have been specified and, in GSM networks, the encryption algorithms are referred to as A5/ x with x taking values from one to four.

A5/2 offered very little security while the limited security offered by A5/1 has been confirmed in a range of papers [10, 22]. Both algorithms were shift register based stream ciphers. In third generation networks, encryption is provided by a different set of algorithms that includes the block cipher KASUMI [2]. Given the existing and widespread deployment of KASUMI in 3G networks it is natural to consider that it be reused in existing 2G networks in the guise of two algorithms called A5/3 and A5/4.

KASUMI is an important block cipher and it has, appropriately enough, been the subject of cryptanalysis over recent years. The elegant design of the algorithm, closely based on MISTY [64], means that the ciphers' resistance to classical block ci-

pher attacks such as differential [13] and linear cryptanalysis [65] is well-understood. However the key schedule of KASUMI offers a very clean framework for analysis and several related-key attacks have been proposed. These began as attacks on reduced-round KASUMI [55, 56], advancing to sophisticated related-key attacks on the full eight rounds of KASUMI [17], and finally culminating in related-key attacks on the full cipher with a time complexity that permitted experimental verification [34].

The purpose of this chapter is to further explore the applicability of related-key attacks to KASUMI. It is well-known, but not always made explicit in cryptanalysis papers, that while related-key attacks on KASUMI are of considerable academic interest they cannot be applied directly within mobile telephony networks. However, there is an additional factor that can limit the applicability of KASUMI related-key attacks and this appears to have been overlooked so far; namely that when KASUMI is used as A5/3 then backward compatibility for session key generation requires that KASUMI be used with 64- and not 128-bit keys. The purpose of this chapter, therefore, is to see whether the related-key attacks in the literature can be applied to KASUMI when used as A5/3.

6.2 KASUMI

The reader should be referred to 2 for details of KASUMI algorithm and notations used in this chapter.

The original basis for *KASUMI* is the block cipher *MISTY* [64]. This was one of the first block ciphers to provide an element of provable resistance to both linear and differential cryptanalysis. While the cipher might at first sight appear to be rather complicated, its innovative recursive design—with a round function that calls upon two mini-Feistel networks FI and FO—is easy to work with and yields

Fig. 6.1: Details of the FL function in the KASUMI cipher. Numbers in brackets indicate the size of the inputs in bits.

useful bounds on the probability and bias of differential and linear characteristics. *KASUMI* inherits all the significant features of the *MISTY* design and the overall form of encryption with *KASUMI* is shown in Figures. 6.2 and 6.3.

The key schedule for *KASUMI* is very simple and is a slightly modified version of the schedule proposed for *MISTY*. The 128-bit user-provided key K is partitioned into 16-bit words $K_0, K_1, \dots, K_7$ and subkeys for round i are given by $KL_i = KL_{i1} \parallel KL_{i2}$, $KO_i = KO_{i1} \parallel KO_{i2} \parallel KO_{i3}$, and $KI_i = KI_{i1} \parallel KI_{i2} \parallel KI_{i3}$. This derivation is illustrated in Figure. 6.4 and Table 6.1.

round	1	2	3	4
KL_{i2}	$K_2 \oplus 89ab$	$K_3 \oplus cdef$	$K_4 \oplus fedc$	$K_5 \oplus ba98$
KI_{i1}	$K_4 \oplus fedc$	$K_5 \oplus ba98$	$K_6 \oplus 7654$	$K_7 \oplus 3210$
KI_{i2}	$K_3 \oplus cdef$	$K_4 \oplus fedc$	$K_5 \oplus ba98$	$K_6 \oplus 7654$
KI_{i3}	$K_7 \oplus 3210$	$K_0 \oplus 0123$	$K_1 \oplus 4567$	$K_2 \oplus 89ab$
round	5	6	7	8
KL_{i2}	$K_6 \oplus 7654$	$K_7 \oplus 3210$	$K_0 \oplus 0123$	$K_1 \oplus 4567$
KI_{i1}	$K_0 \oplus 0123$	$K_1 \oplus 4567$	$K_2 \oplus 89ab$	$K_3 \oplus cdef$
KI_{i2}	$K_7 \oplus 3210$	$K_0 \oplus 0123$	$K_1 \oplus 4567$	$K_2 \oplus 89ab$
KI_{i3}	$K_3 \oplus cdef$	$K_4 \oplus fedc$	$K_5 \oplus ba98$	$K_6 \oplus 7654$

Tab. 6.1: Keys KL_{i2} , KI_{i1} , KI_{i2} , and KI_{i3}

Fig. 6.2: Computational graph for the encryption process of the KASUMI cipher. Details of the function FO is given in Figure. 6.3.

Fig. 6.3: The FO_i function (left) and FI_{ij} function (right) used in round i of the KASUMI cipher. Numbers in brackets indicate the size of the inputs in bits. The 48-bit round key KO_i is interpreted as $KO_{i1}||KO_{i2}||KO_{i3}$ while the 48-bit round key KI_i is interpreted as $KI_{i1}||KI_{i2}||KI_{i3}$. Within FI_{ij} the 16-bit KI_{ij} is interpreted as $KI_{ij1}||KI_{ij2}$.

Fig. 6.4: Details of the key schedule for the KASUMI cipher. The 128-bit user-supplied key is divided into eight 16-bit words, with all the resultant subkeys being generated as a result of bitwise rotations or the addition of a constant. The use of each subkey is illustrated in Figure. 6.2.

6.3 Related Key Attacks and KASUMI

To begin, we follow [52] and establish the basic features of a related-key attack.

Definition 6.3.1. We say that a related key differential $\alpha \rightarrow \beta$ with the key difference ΔK holds for cipher E with probability p , if

$$\Pr_{P,K}[E_K(P) \oplus E_{K \oplus \Delta K}(P \oplus \alpha) = \beta] = p,$$

where $E_K(\cdot)$ denotes encryption through E with the key K . For the ease of exposition, we denote this event by $Pr(\alpha \xrightarrow[\Delta K]{E} \beta)$.

The ΔK that appears in related key attacks offers the attacker considerable power in his cryptanalysis. To see how things change, consider the following two lemmas. The first lemma illustrates the bound on a differential characteristic to the FI function under conventional cryptanalysis; it is stated as a result in [4]. When we allow related key attacks, as in Lemma 6.3.3, the bound is severely degraded. The bounds for both lemmas are tight, i.e., there exist differential characteristics with the stated probability when averaged over all inputs to the function FI.

Lemma 6.3.2. *For any active input difference to the KASUMI function FI with key difference $\Delta(KI) = 0$, the probability of a differential characteristic is $\leq 2^{-14}$.*

Lemma 6.3.3. *For any (active or inactive) input difference to the KASUMI function FI with key difference $\Delta(KI) \neq 0$, the probability of a differential characteristic is $\leq 2^{-6}$.*

Proof. Denote the action of the FI function on text input $\alpha \parallel \beta$, where α and β are 16-bit values, as

$$FI(\alpha \parallel \beta, KI_{i,j,1}, KI_{i,j,2}) = \gamma \parallel \delta.$$

Inspection, see Figure. 6.3, reveals that there are only two instances where a single S-box S7 is differentially active. The first is when

$$\text{FI}(\Delta\alpha \parallel \Delta\beta, \Delta\text{KI}_{i,j,1}, \Delta\text{KI}_{i,j,2}) = 0 \parallel 0$$

with $\Delta\alpha = 0$ and $\Delta\text{KI}_{i,j,2} = \text{ZERO-EXTEND}(\Delta\beta)$. The second is when

$$\text{FI}(0 \parallel 0, \Delta\text{KI}_{i,j,1}, \Delta\text{KI}_{i,j,2}) = \Delta\gamma \parallel \Delta\delta$$

with $\Delta\text{KI}_{i,j,2} = 0$ and $\Delta\delta = \text{ZERO-EXTEND}(\Delta\text{KI}_{i,j,1})$. The result, and the fact that the bound is tight, follows. ■

Already we see the resistance of FI to differential-attack is greatly reduced under related keys.

But there are other reasons to wonder about the potential for related key cryptanalysis against KASUMI. In particular, a glance at the key schedule (see Figure. 6.4) reveals that the process of subkey generation preserves byte boundaries. This means that a difference confined to a single byte of the 128-bit user-supplied key will always lead to a single byte difference in any subsequent subkey in which it appears.

Finally, there is an interesting interaction between related keys and the structure of KASUMI and the FL function. In one way or another, all KASUMI related key attacks in the literature use the following observation which was first used in [23]. Consider a difference in word K_2 of the user-supplied key $\text{K}_0 \parallel \text{K}_1 \parallel \dots \parallel \text{K}_7$. This word of key is used within the FL function in the first round and it is possible that identical round inputs give identical round outputs even when these two different round keys are used. These identical outputs from the FL function are generated

with a probability that depends on the Hamming weight of the key difference¹. See Figure. 6.1 to verify this observation. In the next round, a bitwise rotation of the same key difference is used as the first subkey to the FO function. But with a pair of judiciously-chosen plaintexts, and exploiting the Feistel-like structure of KASUMI, the difference introduced by the subkey can be canceled by an identical difference in the text inputs to FO. In the third round, another bitwise rotation of the difference in subkey K_2 is used as input to the FL function. This is illustrated in Figure. 6.5 and the net result, in typical related key attacks on KASUMI, is that when Δ is chosen as a single-bit difference the three-round characteristic holds, in general, with probability $\frac{1}{4}$.

Fig. 6.5: The basic three-round related key differential for KASUMI with each identified value being 16 bits long. Input keys KO_i to FO are illustrated as inputs from the left while the input keys KI_i come from the right. The probability of the differential for given $x||y$ is $\geq 2^{-2\text{Hwt}(\Delta)}$.

Clearly any three-round differential holding with a probability of at least $\frac{1}{4}$ is going to be very useful in an attack against an eight-round cipher. In a regular related key attack, as explored by Blunden and Escott [23], this good start can be extended at some cost to the probability to give an attack on up to five and six

¹ By fixing the value of bits of the input this probability can be 1 in the first round.

rounds. However we can go further if we turn to some more sophisticated attacks and take further advantage of the structure in KASUMI.

Related Key Boomerang and Rectangle Attacks.

As explained in [52] a *boomerang attack* [93] exploits two short differentials rather than a single longer one. Such an attack is achieved using a combination of chosen plaintext and adaptively chosen ciphertext. Alternatives to this approach, i.e., variants that rely solely on chosen plaintext, have been named the *amplified boomerang attack* [50] and *rectangle attack* [15]. Extending these attacks to a related key setting is, at least in principle, straightforward.

In KASUMI the three-round related key differential running from rounds one to three, see Figure. 6.5, has a close relative in other rounds. A very similar differential holds across rounds four to six when using the same key difference in K_6 . This can be exploited in the reverse direction and, with hindsight, it is now clear that several factors are aligning in the cryptanalysts' favour. In particular we have two high-probability differentials (in the related key model) that cover, together, six out of eight rounds. If we can find a way to define a distinguisher that incorporates one additional round then, in principle, key counting techniques could be used to recover user-supplied key from the entire cipher. The very elegant work of Dunkelman *et al.* [34] provides one way to do this. The “filling” in the so-called *sandwich attack* [34] provides the missing round and allows two three-round related key differentials to be connected, while key-counting on the resultant seven-round distinguisher allows the encryption key to be recovered with very practical resource requirements.

Related Key Attacks in Practice.

At this point it should be observed that the ETSI SAGE designers of KASUMI have always been open in stating that related key attacks are not a particular concern. This is because session keys are derived in a pseudo-random way and it is not possible for an adversary to choose how session keys might be related. Further, KASUMI is specified for use in a mode that generates keystream and this doesn't allow an adversary to specify inputs to the encryption process itself. But even if we were to set aside these operational objections to related key attacks, there is a more fundamental reason for the failure of this attack when using A5/3.

6.4 Related Key Attacks and A5/3

When KASUMI is used as A5/3 it is stipulated that 64-bit keys be used; the associated 128-bit key is obtained by concatenating two copies of the 64-bit key [3]. This restriction on the form of the key is relaxed within the KASUMI-based specifications A5/4, but we are nevertheless left with an interesting question: Given that A5/3 is likely to be an encryption algorithm in widespread use, how do related key attacks on KASUMI translate to this particular choice of algorithm?

6.4.1 Existing Related Key Attacks

The 64-bit key k that is supplied to the KASUMI encryption unit is treated as a 128-bit key of the form $k||k$ [3]. The immediate consequence is that any related key difference gives an active difference in two, four, six, or eight words. These are listed in detail in Table 6.2. It is easy to see the effect of this change on the basic three-round related key differential and this is illustrated in Figure. 6.6. To bound the probability of the differential characteristic now, we can appeal to Lemma 6.3.3

Fig. 6.6: Active key differences for a repeated 64-bit key and the basic three-round related key differential of KASUMI. Keys KO_i are illustrated as inputs from the left while keys KI_i come from the right.

from earlier and a more involved lemma given here:

Lemma 6.4.1. *Write the key inputs to FO as (KO_1, KO_2, KO_3) and (KI_1, KI_2, KI_3) . For any (active or inactive) text input to FO, and for any active key difference in at least one of $(KO_1, KO_2, \text{ or } KO_3)$ there must be at least one FI function that is differentially active except in the following three cases:*

1. $\Delta(KO_1) \neq 0$, $\Delta(KO_2) = 0$, and $\Delta(KO_3) = 0$.
2. $\Delta(KO_1) = 0$, $\Delta(KO_2) \neq 0$, and $\Delta(KO_3) \neq 0$.
3. $\Delta(KO_1) \neq 0$, $\Delta(KO_2) \neq 0$, and $\Delta(KO_3) \neq 0$.

Proof. Cases can be enumerated and evaluated by hand using Figure. 6.3. ■

Lemma 6.4.1 concentrated on how inputs to FO might cause the inner function FI to become differentially active. Of course, independently of whether KO_1 , KO_2 , or KO_3 is active, an FI function can become active directly via differences in the keys (KI_1, KI_2, KI_3) . So an immediate consequence of Lemmas 6.3.3 and 6.4.1 is

the following. The three-round related key differential characteristic that held with probability $\frac{1}{4}$ in KASUMI with 128-bit keys holds with probability $\leq 2^{-18}$ in A5/3. It is unlikely that this bound is tight.

On its own, this result seriously degrades the effectiveness of related key attacks when applied against KASUMI as used in A5/3. The existence of a high probability three-round differential was key to the attack in [34] where, in effect, the differential (and close relatives) are used four times. With the drop in the probability for this three-round differential when moving to 64-bit keys, and since $(2^{-18})^4 = 2^{-72} < 2^{-64}$, we conclude (under typical and reasonable assumptions of independence of differentials) that the specific related key attacks of [34] cannot be mounted against A5/3 without solving some challenging problems. Seeing that existing attacks need a radical rethink is clear; our goal in the next section is to give evidence that the task might be difficult.

6.4.2 Revised Related Key Attacks

In this section we derive an upper bound on the probability of *all* three-round related key differential in A5/3. Since three-round differentials (or longer) feature prominently in all related key attacks on KASUMI, we show that their effectiveness is severely reduced in the context of A5/3. For the particularly efficient attacks described in [34], this new bound on any three-round related key differential is sufficient to demonstrate that the attack, as described in [34], cannot be currently applied to A5/3.

Lemma 6.4.2. *In a round of KASUMI, if FO has one active Δ KI then the maximum probability of a differential characteristic is 2^{-6} . If there are at least two active Δ KI then the maximum probability of a differential characteristic is 2^{-12} .*

Proof. In round i , i odd, we have for $\Delta K \neq 0$, that

$$\begin{aligned} \Pr.(\alpha \xrightarrow{i} \beta) &= \sum_{X=0}^{2^{32}} \Pr.(\alpha \xrightarrow{FL} X) \cdot \Pr.(X \xrightarrow{FO} \beta) \\ &\leq \Pr.(\xrightarrow{FO}_{max} \beta) \sum_{X=0}^{2^{32}} \Pr.(\alpha \xrightarrow{FL} X) \\ &\leq \Pr.(\xrightarrow{FO}_{max} \beta), \end{aligned}$$

where α and β are the input and output difference to the i -th round and X is any given output difference of FL. Similar reasoning holds for i even with X being the output difference from FO. The result now follows since, by Lemma 6.3.3, $\Pr(\xrightarrow{FO}_{max} \beta) \leq 2^{-6}$ when one FI is differentially active and $\leq 2^{-12}$ when two FI functions are differentially active. ■

Lemma 6.4.3. *For any three-round differential of KASUMI across rounds i , $i+1$, and $i+2$, the probability of the differential (in a related key setting) is upper-bounded by*

$$\min\{\Pr.(\Delta^i)_{max} \times \Pr.(\Delta^{i+1})_{max}, \Pr.(\Delta^{i+1})_{max} \times \Pr.(\Delta^{i+2})_{max}, \Pr.(\Delta^i)_{max} \times \Pr.(\Delta^{i+2})_{max}\},$$

where $\Pr.(\Delta^i)_{max}$ denotes the maximum probability of any non-trivial differential characteristic across round i in the related key setting.

Proof. Consider three consecutive rounds of KASUMI and, to give additional precision, denote the input to the three-round differential by $\alpha_L \parallel \alpha_R$ and the output after three rounds, without a swap in the third round, as $\beta_L \parallel \beta_R$. Each of these indicated quantities is 32 bits in length. Then we wish to establish an upper-bound on the probability of the three round differential

$$\Pr.(\alpha_L \parallel \alpha_R \xrightarrow{i, i+1, i+2} \beta_L \parallel \beta_R)$$

round	$\{k_0, k_4\}$	$\{k_1, k_5\}$	$\{k_2, k_6\}$	$\{k_3, k_7\}$
1	KI ₁	KO ₁ , KO ₂	KO ₃	KI ₂ , KI ₃
2	KI ₂ , KI ₃	KI ₁	KO ₁ , KO ₂	KO ₃
3	KO ₃	KI ₂ , KI ₃	KI ₁	KO ₁ , KO ₂
4	KO ₁ , KO ₂	KO ₃	KI ₂ , KI ₃	KI ₁
5	KI ₁	KO ₁ , KO ₂	KO ₃	KI ₂ , KI ₃
6	KI ₂ , KI ₃	KI ₁	KO ₁ , KO ₂	KO ₃
7	KO ₃	KI ₂ , KI ₃	KI ₁	KO ₁ , KO ₂
8	KO ₁ , KO ₂	KO ₃	KI ₂ , KI ₃	KI ₁

Tab. 6.2: Key differences in the 64-bit user-supplied key lead to *at least* the above-noted subkeys being differentially active in the specified round.

in the related key setting. For a given $\Delta K \neq 0$, we have

$$\Pr.(\alpha_L \parallel \alpha_R \xrightarrow{i, i+1, i+2} \beta_L \parallel \beta_R) = \sum_{X=0}^{2^{32}-1} \Pr.(\alpha_L \xrightarrow{i} X) \cdot \Pr.(\alpha_R \oplus X^{i+1} \alpha_L \oplus \beta_L) \cdot \Pr.(\beta_L \xrightarrow{i+2} \alpha_R \oplus X \oplus \beta_R),$$

where X is the output difference of FO in round i for i odd and the output difference of FL in round i for i even. Without loss of generality $\Pr.(\alpha_L \xrightarrow{i} X) < \Pr.(\Delta_{max}^i)$ and $\Pr.(\alpha_R \oplus X \xrightarrow{i+1} \alpha_L \oplus \beta_L) < \Pr.(\Delta_{max}^{i+1})$ for any X . So we have that

$$\begin{aligned} & \sum_{X=0}^{2^{32}-1} \Pr.(\alpha_L \xrightarrow{i} X) \cdot \Pr.(\alpha_R \oplus X \xrightarrow{i+1} \alpha_L \oplus \beta_L) \cdot \Pr.(\beta_L \xrightarrow{i+2} \alpha_R \oplus X \oplus \beta_R) \\ & \leq \Pr.(\Delta_{max}^i) \times \Pr.(\Delta_{max}^{i+1}) \times \sum_{X=0}^{2^{32}-1} \Pr.(\alpha_L \xrightarrow{i} X) \\ & \leq \Pr.(\Delta_{max}^i) \times \Pr.(\Delta_{max}^{i+1}). \end{aligned}$$

The other two cases are treated similarly and the result follows. ■

Theorem 6.4.4. *The probability of any three-round related key differential over*

KASUMI, when used as A5/3, is $\leq 2^{-18}$.

Proof. We appeal to Lemmas 6.4.1, 6.3.3, 6.4.2, and 6.4.3. First we construct Table 6.2 where we note that, due to rotational symmetries in the way subkeys are used, it suffices to consider the first three rounds only. There are 15 cases to consider, depending on which pairs $\{k_0, k_4\}$, $\{k_1, k_5\}$, $\{k_2, k_6\}$, or $\{k_3, k_7\}$ are active. However these are easily broken down into a few cases and enumerated.

- If either of the pairs $\{k_0, k_4\}$ or $\{k_1, k_5\}$ are active, then the result follows from Lemmas 6.3.3, 6.4.2, and 6.4.3.
- If the pair $\{k_2, k_6\}$ or $\{k_3, k_7\}$ are active then the result follows from Lemmas 6.3.3, 6.4.1, 6.4.2, and 6.4.3. ■

The results in this section show that any three-round differential for A5/3, even in the related key model, must have a probability that is upper-bounded by 2^{-18} . Since success in the work of [34] relies on the probability of a three-round differential being raised to the fourth power, and since this probability will be $\leq 2^{-72} < 2^{-64}$, it seems that existing related key techniques cannot be used against A5/3 without potentially significant adjustment.

Note that we have not gone to great lengths to reduce the upper bound beyond what we need. Instead, for clarity, we have restrained ourselves to what we can achieve in a straightforward manner. The bound therefore is unlikely to be tight and there may be considerable scope for improvement.

6.4.3 Implications and Observations

One conclusion from the work in this chapter is that the redundancy in creating a 128-bit key from a duplicated 64-bit key can be very useful, particularly in the

related key setting where any differences in the user-supplied key are immediately amplified. Certainly it seems that it would be very difficult to patch existing related key attacks on KASUMI to give attacks that are anyway near as effective on A5/3 as those that exploit full 128-bit keys, particularly since the work effort in attacking A5/3 would be limited to 2^{64} operations.

The design of key schedules for block ciphers is notoriously *ad hoc* and while there are some guiding principles, such as to use as much of the user-supplied key as possible in different parts of the encryption process (and to do so often), there remain a very broad range of proposals in the literature from the ultra-lightweight [8] to the computationally demanding [83]. Given the interaction between hash function and block cipher design and analysis, and observing a range of new proposals that incorporate some measure of resistance to related key attacks, *e.g.* [41], it seems that an improved understanding of the role of the key schedule would be very useful to designers and analysts alike.

6.5 Conclusion

While KASUMI is proposed for use as A5/3, an important operational restriction lends, perhaps ironically, some additional cryptographic protection. The restriction is that the user-supplied key in A5/3 is 64 and not 128 bits long and this forces redundancy into the typical KASUMI key schedule in the form of repeated key material. For related key attacks this can be a severe drawback since the cryptanalyst would like to avoid the proliferation of differences during the encryption process, be they from the text or the key.

In this chapter we have considered recent related key attacks on KASUMI and demonstrated that for A5/3 they do not apply. It is an open question whether they

can be patched and re-applied to A5/3. It is also unclear whether this duplication of key material might open other opportunities to the attacker. Nevertheless, we have used the elegant structure of MISTY/KASUMI to derive upper-bounds on all three-round differentials for A5/3 in the related key setting, and these bounds indicate that developing a patch might not be straightforward.

7. IMPROVED DIFFERENTIAL CACHE TRACE ANALYSIS ON SMS4

At CT-RSA 2011, the Differential Cache Trace Attacks (DCTA) was introduced and applied to block cipher *CLEFIA* with a cache line size of 32 bytes. In this chapter, we improve the algorithm in term of methodology by efficiently exploiting information of the model of attack. Therefore the uncertainty of secret key and the number of samples required for attack are reduced. A method of choosing samples and several cryptanalytic techniques are developed to make the improved algorithm applicable to *SMS4* block cipher with $2^{3.5}$ to $2^{9.7}$ samples required even in a very large cache size, i.e., 64 or 128 bytes. Especially, we show that the case of 128-byte cache line is the most vulnerable to the attack. The adversary can maximise exploiting the information of the model by using additional information which only occurs in 128-byte cache line and then number of samples required is very few as 11 samples only. The experimental result of timing attack on *SMS4* based on this method is also provided. A simple countermeasure based on key whitening is suggested for *SMS4* cipher from this attack and still valid for the other ciphers like *CAMELLIA*, *CLEFIA*, and *MISTY*.

This chapter is based on the joint work with Chester Reibero, Prof Debdeep Mukhopadhyay and Prof Wang Huaxiong. The experimental results was done by Chester Reibero. Thank you Prof Debdeep Mukhopadhyay and Prof Wang Huaxiong gave us the precious comments for editing the chapter.

7.1 Cache Attacks

Cache attacks are a class of side-channel attacks that target implementations of ciphers that use lookup tables. They were first prophesised by Kelsey et al. in [51] and then theoretically modeled by Page in [77]. In [89] and [90], the first cache attacks were successfully demonstrated on *MISTY1*, *DES*, and *3 – DES*. Since then there have been several variants of cache attacks that either used new attack techniques or enhanced previous attacks.

Cache attacks can be classified into three categories depending on the method of acquiring side-channel information. These are *timing*, *access*, and *trace*. Attacks based on traces [9, 12, 39, 40, 77, 81] make use of power measurements or electromagnetic analysis to gain information about the memory accesses made by the cipher. These attack make use of the fact that a cache miss takes more power and time compared to a cache hit. To improve the speed of the encryption, the data can be stored in cache memory. If the data which is required for the computation is already in cache memory, then we call this event cache hit. If the wanted data is not in cache memory, then the wanted data will be copied from ram memory to cache memory before sending to the computation. This event is called cache miss. Works such as [12] and [81] show that even with a single power trace it is possible to extract sufficiently large amounts of information about the memory accesses.

For ciphers having a Feistel structure, a trace attack that utilized the differential properties of the S-box was proposed in [81] and used to attack *CLEFIA*. The attack, which is known as DCTA, was improved in [82]. DCTA was also applied to the *CAMELLIA* block cipher in [78]. In the present chapter we attack the *SMS4* cipher using DCTA. The *SMS4* cipher [5] is used in the WAPI (WLAN Authentication and Privacy Infrastructure) standard to protect WLAN products in

China. *SMS4* has a Feistel structure but differs from *CLEFIA* and *CAMELLIA* in that it has a type-1 generalized structure[91], compared to the type-2 structure[91] in *CLEFIA* and the standard Feistel structure in *CAMELLIA*. In this chapter we show that cipher's with a type-1 structure are prone to a more powerful DCTA, which exploit the structure properties. The attack on *SMS4* presented in this chapter, not only reduces the number of samples required to implement the attack, but also the uncertainty of the remaining key space. On a standard 64 byte cache line, the number of plaintexts required is $2^{7.7}$, which reduces the key space from 2^{128} to 2^{24} . On 128 byte cache lines, $2^{3.5}$ (11) plaintexts is capable of reducing the key space to 2^{28} . The contributions of the chapter are as follows:

1. We develop a method of choosing samples for *SMS4* due to which the number of samples required is drastically reduced.
2. We show that the adversary can maximize the efficiency of the attack with 1-bit collision by reusing the samples and exploiting more information from the cache miss. Hence the number of plaintexts needed is only 11 samples in case of 128-byte cache.
3. Key-Whitening can be used as a very simple method to protect the master key and as we know this is the first time this idea is applied.
4. The complexity of the attack is analyzed for several cache line sizes and experimental result of timing attack is presented.

The rest of the chapter is organized as follows. Section 7.2 summarizes the idea of DCTA. In Section 7.3, the description of *SMS4* block cipher is provided. Then in Section 7.4 we describe the improved attacks on *SMS4* in details. We show how to recover the secret key, the improvement of the methodology of attack. Then we

move to section of protection method by using the key-whitening. In Section 7.6, all the computed parameters used in attack which have to be determined based on experiments are described and we present the experimental result of timing attack on SMS4. The last section we conclude the chapter.

7.2 Differential Cache Trace Attacks

In [81] the DCTA was introduced for ciphers with the Feistel structure. We follow the explanation in [81] in order to introduce the attack technique.

Consider a 2-round Feistel cipher as shown in Figure 7.1. There are 2-keys k_0 and k_1 each of n bits, and two inputs in_0 and in_1 . The S-box has 2^l elements implemented by a lookup table. Assume that 2^m elements of the table share a cache line. This means that the table occupies 2^{l-m} cache lines. Now, suppose in_1 is chosen such that the second S-box access collides with the first then,

$$\langle in_0 \oplus k_0 \rangle = \langle S[in_0 \oplus k_0] \oplus in_1 \oplus k_1 \rangle, \quad (7.1)$$

where $\langle \cdot \rangle$ denotes $(l - m)$ most significant bits. This results in the reduction of the uncertainty for k_0 and k_1 from 2^{2l} to 2^{l+m} . Next, suppose in'_0, in'_1 is another pair satisfying the Equation (7.1), then a difference equation can be derived as follows:

$$\langle in_0 \oplus in_1 \oplus in'_0 \oplus in'_1 \rangle = \langle S[in_0 \oplus k_0] \oplus S[in'_0 \oplus k_0] \rangle. \quad (7.2)$$

Based on the property of S-box S , the key k_0 can be uniquely recovered by using Equation (7.2) with a certain number of pairs of (in_0, in_1) . We denote the number of inputs required by N_m . The value of N_m strongly depends on S-box S and the size of the cache line. In this chapter, we show how carefully chosen values for the

Fig. 7.1: Two Round Feistel Structure [81].

inputs can be used to reduce the uncertainty of the keys further.

7.3 The SMS4 Block Cipher

The reader is directed to 2 for notations and description of *SMS4* algorithm which are needed in this chapter. The Figure 7.2 describes the first 4 rounds of *SMS4* which are considered in this chapter.

7.4 The Improved Differential Cache Trace Attack on SMS4

From the key schedule it can be seen that the entire 128 bit *SMS4* secret key can be uniquely recovered from four round keys RK_0 , RK_1 , RK_2 , and RK_3 [32]. In this section we describe the methodology of our attack on *SMS4*. We assume that the cache line is of size m bytes, and the table used to implement the S-box is of 2^l bytes. The number of most significant bits revealed from each cache access is therefore $n = l - \log_2 m$. We denote these n most significant bits by $\langle \cdot \rangle$. In the remainder of this section we show how these n bits can be used to reveal round keys

Fig. 7.2: First 4 rounds of SMS4.

RK_0 , RK_1 , RK_2 , and RK_3 , thus recovering the SMS4 secret key.

7.4.1 Step 1: Recovering RK_0

Let a plaintext (X_0, X_1, X_2, X_3) be chosen such that there are 7 cache hits in first 2 rounds. The following equations can then be derived:

For Round 0:

$$\begin{aligned}
 \langle X_{1,0} \oplus X_{2,0} \oplus X_{3,0} \oplus RK_{0,0} \rangle &= \langle X_{1,1} \oplus X_{2,1} \oplus X_{3,1} \oplus RK_{0,1} \rangle, \\
 &= \langle X_{1,2} \oplus X_{2,2} \oplus X_{3,2} \oplus RK_{0,2} \rangle, \\
 &= \langle X_{1,3} \oplus X_{2,3} \oplus X_{3,3} \oplus RK_{0,3} \rangle.
 \end{aligned} \tag{7.3}$$

For Round 1:

$$\begin{aligned}
& \langle X_{1,0} \oplus X_{2,0} \oplus X_{3,0} \oplus RK_{0,0} \rangle \\
&= \langle X_{0,0} \oplus X_{2,0} \oplus X_{3,0} \oplus RK_{1,0} \oplus T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)_0 \rangle, \\
&= \langle X_{0,1} \oplus X_{2,1} \oplus X_{3,1} \oplus RK_{1,1} \oplus T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)_1 \rangle, \quad (7.4) \\
&= \langle X_{0,2} \oplus X_{2,2} \oplus X_{3,2} \oplus RK_{1,2} \oplus T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)_2 \rangle, \\
&= \langle X_{0,3} \oplus X_{2,3} \oplus X_{3,3} \oplus RK_{1,3} \oplus T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)_3 \rangle.
\end{aligned}$$

Such a plaintext can be chosen adaptively as follows. For round 0, choose arbitrary values of X_0 , X_2 , and X_3 , and $X_{1,0}$, $X_{1,2}$, $X_{1,3}$. First vary $X_{1,1}$ until a cache hit is obtained. Then vary $X_{1,2}$ until the second cache hit is obtained, and finally $X_{1,3}$ till the third cache hit is obtained. In a similar way 4 cache hits can be obtained in the second round by varying each byte of X_0 one after the other.

Let P, P' be two such plaintexts which satisfy Equations 7.3 and 7.4, then we have the following equations:

$$\begin{aligned}
\langle RK_{0,1} \rangle &= \langle X_{1,1} \oplus X_{2,1} \oplus X_{3,1} \oplus X_{1,0} \oplus X_{2,0} \oplus X_{3,0} \oplus RK_{0,0} \rangle, \\
\langle RK_{0,2} \rangle &= \langle X_{1,2} \oplus X_{2,2} \oplus X_{3,2} \oplus X_{1,0} \oplus X_{2,0} \oplus X_{3,0} \oplus RK_{0,0} \rangle, \quad (7.5) \\
\langle RK_{0,3} \rangle &= \langle X_{1,3} \oplus X_{2,3} \oplus X_{3,3} \oplus X_{1,0} \oplus X_{2,0} \oplus X_{3,0} \oplus RK_{0,0} \rangle.
\end{aligned}$$

$$\begin{aligned}
& \langle X_{1,0} \oplus X_{0,0} \oplus X'_{1,0} \oplus X'_{0,0} \rangle \\
& \quad = \langle T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)_0 \oplus T(RK_0 \oplus X'_1 \oplus X'_2 \oplus X'_3)_0 \rangle, \\
& \langle X_{1,1} \oplus X_{0,1} \oplus X'_{1,1} \oplus X'_{0,1} \rangle \\
& \quad = \langle T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)_1 \oplus T(RK_0 \oplus X'_1 \oplus X'_2 \oplus X'_3)_1 \rangle, \\
& \langle X_{1,2} \oplus X_{0,2} \oplus X'_{1,2} \oplus X'_{0,2} \rangle \\
& \quad = \langle T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)_2 \oplus T(RK_0 \oplus X'_1 \oplus X'_2 \oplus X'_3)_2 \rangle, \\
& \langle X_{1,3} \oplus X_{0,3} \oplus X'_{1,3} \oplus X'_{0,3} \rangle \\
& \quad = \langle T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)_3 \oplus T(RK_0 \oplus X'_1 \oplus X'_2 \oplus X'_3)_3 \rangle.
\end{aligned} \tag{7.6}$$

Using Equation 7.5, the uncertainty of key RK_0 reduces from 2^{32} to 2^{32-3n} . Equation 7.6 can be used to further filter the candidates for RK_0 . To determine uniquely the correct value for RK_0 , we need to repeat the filtering for several pairs of plaintexts. Define N_{m_0} as the number of plaintexts that are required to uniquely determine the key. The value of N_{m_0} depends on the S-box and diffusion property of the T function and the cache line size (m). If m increases then N_{m_0} also increases, as the bits revealed from the cache hits reduces. Notably, the Equation 7.6 can be considered as $4n$ -bit filter, i.e., if the length of the RK_0 is $32 - 3n$ bits, i.e., there are 2^{32-3n} key candidates, then after using one pair of plaintexts, the number of remaining key candidates would be $2^{(32-3n)/4n}$. Hence, the theoretical formula of $N_{m_0} \approx (32 - 3n)/4n$. Since, the Equation 7.6 is constructed from differential equations, N_{m_0} is required as least 2, i.e., $N_{m_0} \geq 2$

If the size of the table is 256 bytes, then the number of plaintexts required to uniquely determine RK_0 is $7N_{m_0}2^n$. Note that this approach is different from the previous attacks in [78, 81, 82], because it retrieves the entire 32 bits of the round

key simultaneously, while the previous attacks targeted 8 bits at a time. Targeting 8 bits at a time would require several more plaintexts to uniquely determine the entire round key. Further, during the process of recovering RK_0 , there are $4n$ bits of RK_1 that are also revealed (i.e., $\langle RK_{1,0} \rangle, \langle RK_{1,1} \rangle, \langle RK_{1,2} \rangle, \langle RK_{1,3} \rangle$), this is utilized in the next step of the attack.

7.4.2 Step 2: Recovering RK_1

A straight forward application of the technique in [81] to obtain RK_1 requires choosing plaintexts such that there are 11 cache hits in the first 3 rounds. We note that X_1, X_2 , and X_3 contribute in all 3 rounds, and (as seen in the earlier step to find RK_0), X_0 and X_1 are chosen based on X_2 and X_3 . This *dependence* between the input words results in large number of plaintext required. A naïve technique to obtain cache hits in the third round requires to checks all 2^{32} candidates of X_2 thus resulting in $7N_{m_2}2^{32+n}$ plaintexts. Unless an alternate method of choosing samples is developed, it cannot be directly applied to *SMS4*. Our approach is to make the process of choosing X_0, X_1, X_2 , and X_3 independent of each other. That is the choice of the value of X_i is not based on the choice of the value of X_j , $i, j = \{0, 1, 2\}$.

The method requires $X_2 = X_3$ and 11 cache hits in the first 3 rounds, i.e., rounds 0, 1, 2. By setting $X_2 = X_3$, the value $X_2 \oplus X_3$ is nullified in Round 0 and Round 1. This makes the procedure of choosing samples that collide in Round 0, 1, 2 independent. Cache hits in round 0 can be obtained by using X_1 , while X_0 can be used to obtain cache hits in round 1. For round 2, X_3 can be used. When 11 cache hits are obtained in the first three round, the following equations can be deduced.

For round 0:

$$\begin{aligned}
 \langle X_{1,0} \oplus RK_{0,0} \rangle &= \langle X_{1,1} \oplus RK_{0,1} \rangle, \\
 &= \langle X_{1,2} \oplus RK_{0,2} \rangle, \\
 &= \langle X_{1,3} \oplus RK_{0,3} \rangle.
 \end{aligned} \tag{7.7}$$

For Round 1:

$$\begin{aligned}
 \langle X_{1,0} \oplus RK_{0,0} \rangle &= \langle X_{0,0} \oplus RK_{1,0} \oplus T(RK_0 \oplus X_1)_0 \rangle, \\
 &= \langle X_{0,1} \oplus RK_{1,1} \oplus T(RK_0 \oplus X_1)_1 \rangle, \\
 &= \langle X_{0,2} \oplus RK_{1,2} \oplus T(RK_0 \oplus X_1)_2 \rangle, \\
 &= \langle X_{0,3} \oplus RK_{1,3} \oplus T(RK_0 \oplus X_1)_3 \rangle.
 \end{aligned} \tag{7.8}$$

For round 2:

$$\begin{aligned}
 \langle X_{1,0} \oplus RK_{0,0} \rangle &= \langle X_{3,0} \oplus X_{1,0} \oplus X_{0,0} \oplus T(RK_0 \oplus X_1)_0 \oplus RK_{2,0} \oplus T(RK_1 \oplus X_0 \oplus T(RK_0 \oplus X_1))_0 \rangle, \\
 &= \langle X_{3,1} \oplus X_{1,1} \oplus X_{0,1} \oplus T(RK_0 \oplus X_1)_1 \oplus RK_{2,1} \oplus T(RK_1 \oplus X_0 \oplus T(RK_0 \oplus X_1))_1 \rangle, \\
 &= \langle X_{3,2} \oplus X_{1,2} \oplus X_{0,2} \oplus T(RK_0 \oplus X_1)_2 \oplus RK_{2,2} \oplus T(RK_1 \oplus X_0 \oplus T(RK_0 \oplus X_1))_2 \rangle, \\
 &= \langle X_{3,3} \oplus X_{1,3} \oplus X_{0,3} \oplus T(RK_0 \oplus X_1)_3 \oplus RK_{2,3} \oplus T(RK_1 \oplus X_0 \oplus T(RK_0 \oplus X_1))_3 \rangle.
 \end{aligned} \tag{7.9}$$

RK_1 can be recovered using Equations 7.7 and 7.9 and following a procedure similar to RK_0 . Additionally $4n$ bits of RK_2 are also obtained. The number of additional plaintexts required to retrieve RK_1 is $3N_{m_1}2^n$, where $N_{m_1} \approx (32 - 4n)/4n$ and $N_{m_1} \geq 2$.

7.4.3 Step 3: Recovering RK_2

In order to recover RK_2 , collisions are required in Rounds 2 and 3. However it can be seen that obtaining collisions in Round 3 will disturb the collision in the previous rounds, thus resulting in ambiguous equations. The previous works in [78, 81, 82] applied methods that resulted in probabilistic outputs in many cases. For *SMS4* we show that the structure of the cipher can be exploited in order to obtain deterministic results. Our method introduces a way of choosing samples so as to be able to apply the differential cache techniques to RK_2 .

We will vary $X_{2,0}, X_{2,1}, X_{2,2}, X_{2,3}$ to obtain the cache hits in the Round 3, i.e., four accesses in the round 3 collides with the last S-box access in the Round 2. Since the X_2 is the input of the $T(\cdot)$ function of Round 0 and Round 1, it will disturb the collision in the Round 3. To solve this problem, we set $X_1 = T(RK_1 \oplus X_2 \oplus X_3)$ and $X_0 = T(RK_0 \oplus X_1 \oplus X_2 \oplus X_3)$. Then $X_4 = 0$ and $X_5 = 0$. Hence, X_2 only affects the Round 3 and when four cache hits are obtained in the Round 3, the following equations can be deduced.

$$\begin{aligned}
\langle X_{3,3} \oplus RK_{2,3} \rangle &= \langle X_{2,0} \oplus RK_{3,0} \oplus T(RK_2 \oplus X_3)_0 \rangle, \\
&= \langle X_{2,1} \oplus RK_{3,1} \oplus T(RK_2 \oplus X_3)_1 \rangle, \\
&= \langle X_{2,2} \oplus RK_{3,2} \oplus T(RK_2 \oplus X_3)_2 \rangle, \\
&= \langle X_{2,3} \oplus RK_{3,3} \oplus T(RK_2 \oplus X_3)_3 \rangle.
\end{aligned} \tag{7.10}$$

RK_2 can be recovered using Equation 7.10 and following a procedure similar to RK_0 . Additionally $4n$ bits of RK_3 are also obtained. The number of additional plaintexts required to retrieve RK_2 is $3N_{m_2}2^n$, where $N_{m_2} \approx (32 - 4n)/4n$ and $N_{m_2} \geq 2$ ($N_{m_2} = N_{m_1}$).

7.4.4 Step 4: Recovering RK_3

During the process of find RK_2 we have also obtained $4n$ bits of RK_3 . The remaining bits of RK_3 is found by brute force. For this we require one valid pair of plaintext and ciphertext. Then, for every guessed value of RK_3 (there are 2^{32-4n} guesses), we reverse the key scheduling to obtain a candidate key. The candidate key is correct if the ciphertext obtained by encrypting the given plaintext with the candidate key results in the correct ciphertext.

7.4.5 The Theoretical Number of Pairs of Samples Needed

Denote N the total number of pairs of samples needed. Then,

$$\begin{aligned} N &= (7N_{m_0} + 3N_{m_1} + 3N_{m_2})2^n \\ &\approx 13\lceil N_{m_0} \rceil 2^n. \end{aligned} \tag{7.11}$$

We compute the theoretical values of N_{m_0} and $N_{m_1}(=N_{m_2})$ and N in Table 7.1.

Number of Collision - $n = 8 - \log_2 m$	1 bit	2 bits	3 bits	4 bits	5 bits
N_{m_0}	7.25	3.25	2	2	2
N_{m_1}	7	3	2	2	2
N	208	208	208	416	832
$\log_2(N)$	7,7	7,7	7,7	8,7	9,7

Tab. 7.1: Theoretical Complexity of Attack vs Cache Line Size.

7.4.6 The Special Attack when $n = 1$

Although this method can be tailored for any value of n , it is most suited for $n = 1$, ie. 128 byte cache lines, because of the following lemma.

Lemma 7.4.1. *In case of 128-byte cache line, the cache miss and hit provide the same information for the attack due to the size of collision is 1 bit.*

According to the lemma, for a given sample, the Equations 7.7, 7.8, 7.9, and 7.10 can be constructed by observing the cache miss and cache hit, then adding one in case of a cache miss would give the required equation. By using this method, the estimated total number of pairs to recover RK_0 , RK_1 and RK_2 reduces from 208 to 8. Since, with a given N_p plaintexts, there are at least $N_p - 1$ pairs of plaintexts and at most $C_{N_p}^2$ pairs, the number of plaintexts needed N_p is less or equal to $N + 1$. In all, the number of plaintexts to recover all there keys (RK_0 , RK_1 , and RK_2) is around 9.

7.5 Protection method against the Cache Attacks based on Key-Whitening

Many countermeasures are proposed to prevent a given cipher from this attack. All the countermeasures share the same method: try to eliminate the dependence of the cache load on the value of key. This can be done by disabling the cache or constructing a table look up small enough to put in a single cache line. The speed of encryption of cipher would dramatically drop without the cache. Constructing a small table lookup is required to use the logical equations for the S-box and this can be faced with another threat such as side-channel attack based on glitches. Then, to construct a secure table lookup, the countermeasure for protecting the table lookup would use many hardware resource. Hence, the size of the constructed table lookup is over room of the cache.

Proposed method: The method is based on the following observations on the attack algorithm:

1. All the round keys are recovered independently.
2. The master key K is computed by using several consecutive round keys.
3. The master key K need to be protected not the values of round keys.

Our idea is even the adversaries can recover some round keys by using this attack, they still can not recover the master key K . This idea can be realized by using a very simple trick: the values of subkeys generated from the master key K which are able to be used for reversely computing master key K are not revealed even in the case of that round keys are recovered by using this attack. This can be done by using the key whitening. To clarify the method, we come back to *SMS4* case.

The master key K of *SMS4* is able to be reversely computed from RK_0, RK_1, RK_2, RK_3 and it generates 32 subkeys $RK_0, \dots, RK_{31}$. These subkeys $RK_0, \dots, RK_{31}$ are used as round keys. Now, we use the master key K to generate 64 subkeys $RK_0, \dots, RK_{63}$ instead of 32. Then, the 32 following values are used as the round keys $RK_0 \oplus RK_{32}, \dots, RK_{31} \oplus RK_{63}$. The adversary can apply the attack to recover the values $RK_0 \oplus RK_{32}, \dots, RK_3 \oplus RK_{35}$ but the values of subkeys $RK_0, \dots, RK_3$ are still not revealed. The adversary has to exhaustive search 2^{128} key candidates if he wants to recover $RK_0, \dots, RK_3$ by using the recovered values of round keys. Hence, the master key K is protected from the attack. This method is still valid for protecting the vulnerable ciphers *CAMELLIA*, *CLEFIA*, and *MISTY*.

This simple method only requires to double the number of subkeys generated from the master key K . All the structure of cipher is not changed. It means that all the cryptographic properties/degree of security of the cipher are preserved.

In case of that many plaintexts are encrypted by using one secret key, we can save much time by using this method in comparison to the previous countermeasures.

Note that, the cipher is changed already because the key schedule is modified to resist DCA.

7.6 Analysis and Experimental Evaluation

To practically mount the attack, the adversary requires to distinguish between a cache hit and a miss. There are two ways this can be done: either using power (or electro-magnetic) traces or by timing the encryption. A single trace can reveal significant information about the cache access patterns. For example in [81], a single power trace was used to reveal all cache access patterns in a round simultaneously.

We analyze the attack algorithm by simulating the attack with various cache line sizes. The aim of the analysis was to estimate the number of plaintexts required for the attack in practice (i.e., the practical $\lceil N_{m_0} \rceil$ and N), and also the number of candidate keys remaining at the end of the attack. Table 7.2 presents the analysis assuming that the table occupies 256 bytes.

Number of Collision - $n = 8 - \log_2 m$	1 bit	2 bits	3 bits	4 bits	5 bits
N_{m_0}	10	4	2	2	2
$\log_2(\text{Number of plaintexts needed})$	3.5	7.7	7.7	8.7	9.7
$\log_2(\text{Candidate Keys of } RK_3)$	28	24	20	16	12

Tab. 7.2: Attack Complexity vs Cache Line Size.

7.6.1 The Timing Attack Algorithm

Attacks that use timing are not so straightforward. Several timing measurements are required in order to distinguish a single memory access. The advantage of the timing side-channel is that it can be mounted remotely thereby resulting in attacks over a network. Now we present the general procedure for a timing attack and then its application to SMS4.

The cipher is considered as a series of r memory accesses to the S-box table. Out of these memory accesses, one (called M_i , $0 \leq i < r - 1$) is kept constant and another (called M_j , $i < j \leq r - 1$) is traversed through every possible cache line that the table can occupy. For example if the table has 2^l elements and the size of the cache line is m , then the most significant bits of M_j is traversed from 0 to $\frac{2^l}{m} - 1$. For a constant M_i and each possible value taken by M_j , the remaining memory accesses are made to random locations until a distribution of the timing can be built. We thus obtain $\frac{2^l}{m}$ distributions of which the distribution with the minimum average time is a cache hit. The details of this is presented in Algorithm 11

Input: memory accesses M_i and M_j , where $0 \leq i < j \leq r$.
Output: The expected value of M_j .
begin
 Choose values of plaintext such that M_i is a constant. ;
 for $s \in \{0, 1, 2, \dots, \frac{2^l}{m} - 1\}$ **do**
 for *several number of times* **do**
 1. Choose values of plaintext such that $M_j = s$.
 2. Choose plaintext so that remaining accesses are random.
 3. Trigger encryption and obtain the time required.
 end
 end
 return *The value of s with the maximum average time.*
end

Algorithm 11: Algorithm to Find Collision.

The reason the algorithm works is that in the memory access M_j , there is exactly one cache line which results in a collision with M_i in every encryption. This therefore results in a distribution which is visibly different compared to the other distributions.

7.6.2 Application to SMS4

We now apply Algorithm 11 to the various steps of the attack.

Determining RK0, RK1 and RK2 : For *RK0* Equations 7.3 and 7.4 needs to be determined, for *RK1*, Equations 7.7, 7.8, and 7.9 need to be determined, while for *RK2*, Equation 7.10 need to be determined. This is done by using the plaintexts on the left hand side of each equation as M_i , and the right hand side as M_j . For example M_i is $X_{1,0} \oplus X_{2,0} \oplus X_{3,0}$ and M_j is $X_{1,1} \oplus X_{2,1} \oplus X_{3,1}$, then the first equality in Equation 7.3 is obtained. In a similar manner the other collisions can be determined.

7.6.3 Experimental Results

We use an Intel Xeon (E5606) as a target platform for the attack. The Xeon has a 32KB L1 data cache which is 8 way set associative. The unified L2 cache is 256KB and is also 8 way associative. There is also an L3 cache of 8MB which is 16 way set associative. The cache line size in all cases in 64 bytes.

The *SMS4* implementation which was attacked was written in C and used a single table of 256 bytes to implement the S-box. Thus $2^l = 256$, $m = 64$, and $n = 2$. In order to distinguish a collision, several encryptions are required in order to build the timing distribution. The error in detection varies inversely with the number of measurements made. Figure 7.3 shows the percentage error in detecting collisions in rounds 1, 2, and 3. It can be seen that as the depth of the attack increases, the error increases and it takes more number of measurements to distinguish the collision.

Fig. 7.3: Error in Detecting a Collision vs Number of Encryptions on an Intel Xeon Sever.

7.7 Conclusions

In this chapter, the improved DCTA is studied on *SMS4*. The improved attacks is achieved by exploiting more information received from model of attack which reduces the uncertainty of secret key. Hence, the number of samples required for the attack is reduced. To our best knowledge, this is the first and the best result of DCTA on *SMS4* so far. A method of choosing samples was shown which makes the differential cache trace applicable on *SMS4*. Hence, it gives us a deep insight into the improved DCTA on *SMS4* block cipher. It tells us that, even for a large cache line size (64-, 128-byte cache), the attack is workable with small number of samples required and practical time. Besides that the very simple and efficient method based on key whitening is introduced to boost resistance of attacked ciphers against this attack. The protection method saves much time in comparison to the previous countermeasure in case of that many plaintexts are encrypted by using one

secret key. Furthermore, the experimental result of timing attack is presented.

8. IMPROVEMENTS IN THRESHOLD COUNTERMEASURE

One of the most promising lightweight hardware countermeasures against SCA attacks is the so-called Threshold Implementation (TI) [74] countermeasure. In this work we discuss issues towards its applicability and introduce solutions to boost its implementation efficiency. In particular, our contribution is three-fold: first we introduce two methodologies to efficiently implement 3-share TI to a given S-box. Second, as an example, we successfully apply these methodologies to PRESENT and are able to decrease the area requirements of its protected S-box by 37-40%. Third, we present the first successful practical Mutual Information Attack on the original 3-share TI implementation of PRESENT and compare it with a correlation-enhanced collision attack using second-order moments.

This chapter is based on the joint work with Sebastian Kutzner, Prof Axel Poschmann and Prof Wang Huaxiong. The attack result and implementation of improved Present cipher was done by Sebastian Kutzner and Prof Axel Poschmann, respectively. The result of Horizontal Level is developed through the discussion with Prof Axel Poschmann. Thank you Sebastian Kutzner, Prof Axel Poschmann and Prof Wang Huaxiong for editing this chapter.

8.1 Introduction

Side Channel Attacks (SCA) [53] were introduced in 1997 by Kocher *et al.* and exploit the fact that while a device is processing data, information about this data

is leaked through different channels, e.g., power consumption, electromagnetic emanation and so forth. DPA [54] is a well known technique analyzing multiple measurements with statistical methods. It exploits the correlation between intermediate results, which partly depend on a secret, and the power consumption.

Several countermeasures (far too many to address all of them) have been proposed during the last years, for example, to decrease the SNR ratio [61], to balance the leakage of different values [79] or to break the link between the processed data and the secret, i.e., masking [61]. Due to the presence of glitches, masked implementation might still be vulnerable to DPA [61]. A recent countermeasure against DPA was introduced in 2006 by Nikova *et.al.* [73] and is called *Threshold Implementation* (TI). It is based on secret sharing (or multi-party computation) techniques and is provable secure against first order DPA even in the presence of glitches. Furthermore, it can be implemented very efficiently in hardware [80].

The number of shares required for a TI depends on the degree d of the non-linear function (S-box) and [73, 74] have shown that one needs at least $d + 1$ shares. It implies that the higher the degree of the non-linear function, the more shares are required and the larger the implementation gets. Since a degree of two is the minimal degree of a non-linear function, the optimal number of shares is three. Therefore, to apply a 3-share TI to a larger degree function, this function must be represented as a composition of quadratic functions [80].

This chapter is organized as follows. In Section 8.2, the TI countermeasure is described. We introduce two methodologies to efficiently implement 3-share the TI countermeasure to an S-box (Section 8.3). We then successfully apply these methodologies to the PRESENT S-box, reducing the hardware implementation costs of the S-box layer by 37 – 40% (Section 8.4). Last, we investigate the security of our new design with practical measurements. In addition, we present a successful practical

Mutual Information attack against the original TI implementation of [80] and compare it with a correlation-enhanced collision attack using second-order moments [69] (Section 8.5). Finally, the chapter is concluded in Section 8.6.

8.2 The Threshold Implementation Countermeasure

In this section we recall the preliminaries of the Threshold implementation countermeasure and revisit the results of [80] describing a 3-share TI of *PRESENT*.

8.2.1 Threshold Implementation Countermeasure

In [73], TI was introduced as a side channel attack countermeasure. It is based on secret sharing and multiparty computation and provable secure against the 1-st order DPA, even in the presence of glitches. The method can be described as follows. The variable x is divided into s shares x_i , $1 \leq i \leq s$, such that $x = \bigoplus_{i=1}^s x_i$. Let

$$F(x, y, z, \dots) : GF(2)^m \rightarrow GF(2)^n$$

be a vectorial boolean function which needs to be shared. Denote $\bar{x} = (x_1, x_2, \dots, x_s)$ and $\bar{x}_i = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_s)$, i.e., the vector $\bar{x}_i$ does not contain the share x_i . In order to share F , a set of s vectorial boolean functions F_i is constructed and must fulfill the following properties:

1. **Correctness:** $F(x, y, z, \dots) = \bigoplus_{i=1}^s F_i(\bar{x}_i, \bar{y}_i, \bar{z}_i, \dots)$.
2. **Non-completeness:** All the function F_i must be independent to the input variables $x, y, z, \dots$, i.e., the inputs of F_i does not have $x_i, y_i, z_i, \dots$ or $F_i = F_i(\bar{x}_i)$.

According to the Theorem 2 and 3 of [73], if the inputs satisfy the following condition

$$Pr(\bar{x} = \bar{X}, \bar{y} = \bar{Y}, \dots) = qPr(x = \bigoplus_i^s X_i, y = \bigoplus_i^s Y_i, \dots), \quad (8.1)$$

where q is a constant, then the shared version of function F can resist against the first order DPA in presence of glitches.

In general, the function F is a S-box layer and the output of the previous round of the cipher is the input of the current round. Hence, we have a following property for the output of F . Assume that, the output of F is $u = (u_1, u_2, \dots, u_i)$, then we have the following property for output of F :

Uniformity: $Pr(\bar{u} = \bar{U} | u = \bigoplus_i^s U_i)$ is a constant.

If the function $u = F(x)$ is invertible, then every vector $\bar{u}$ is reached for exactly one input vector $\bar{x}$. In this chapter and the next one, the function F is a 4-bit S-box. Hence, its 3-share version is required to be 12-bit permutation.

The number of shares s depends on the degree of the original vectorial boolean function $F(x, y, z, \dots)$. Assume that the degree of F is d , then s is computed as follows:

Theorem 8.2.1. [74] *The minimum number of shares required to implement a product of d variables satisfying Property 1 and 2 is given by*

$$s \geq 1 + d.$$

Since the minimum degree of a nonlinear vectorial boolean function is 2, the number of shares s is at least 3. The more shares are needed, the bigger the hardware implementation. Therefore, the 3-share TI is the most efficient case.

8.3 One S-box for all

In this section, we introduce two methodologies to improve the hardware implementation costs for TI. To illustrate our contribution we chose PRESENT as an example. Figure 8.1 shows how to apply the TI to a 4-bit S-box: first it is decomposed into two stages $G(\cdot)$ and $F(\cdot)$ (horizontal), then each stage is shared (vertical).

Figure 8.1 also shows that in [80] the authors implemented $F(\cdot)$ and $G(\cdot)$ using *six different* 8×4 vectorial Boolean functions $f_1, f_2, \dots, g_3$. In the following we will show how to implement the same functionality with only *one* 8×4 vectorial Boolean function, which significantly reduces the area/memory requirement of the TI S-box layer.

Fig. 8.1: Decomposition of an S-box [80]

All the 4-bit permutations constitute the symmetric group S_{16} [48]. The identity permutation is an even permutation. An even permutation can be obtained as the composition of an even number and only an even number of exchanges (called transpositions) of two elements, while an odd permutation be obtained by (only) an odd number of transpositions [48]. All 4-bit even permutations in S_{16} constitute a subgroup which called A_{16} . If a permutation belongs to A_{16} , then it has a parity of +1. In another case, it is said that it has a parity of -1. Let B_{16} be the set of all 4-bit odd permutations, i.e., $B_{16} = S_{16} \setminus A_{16}$.

According to [20], only the 4-bit S-boxes in alternating group A_{16} can be de-

composed into quadratic permutations. The interested reader is referred to [20] for more details. [58] proposed a classification of 4-bit S-boxes based on the following relationship:

Definition 8.3.1. [58] Two S-boxes $S(x), S'(x)$ are linearly equivalent if and only if there exist two 4×4 -bit invertible matrices A, B and two 4-bit vectors c, d such that

$$S'(x) = A(S(Bx \oplus c) \oplus d), \forall x \in \{0, \dots, 15\}$$

The definition above shows us that we can convert one S-box to another from the same class simply by using linear transformations. We will use this fact later for optimizing hardware implementations of a given S-box.

8.3.1 The horizontal level

In order to apply the 3-share TI to a cubic S-box $S(\cdot)$, first the S-box is decomposed into a composition of two quadratic permutations $F(\cdot)$ and $G(\cdot)$ (see Figure 8.1).

Observation 8.3.2. Assume a vectorial Boolean function $S(\cdot) = G(G(\cdot))$, where $G(\cdot)$ is a vectorial boolean function. Then the hardware implementation costs of $S(\cdot)$ may be reduced by reusing the implementation of $G(\cdot)$. Experiments have shown that the costs for additional logic, e.g., a multiplexer, is less than implementing $G(\cdot)$ twice. Numbers are provided in Section 8.4.

The main problem of Observation 8.3.2 is how to find a suitable $G(\cdot)$ such that $G(G(\cdot))$ belongs to the desired class, e.g., class C_{266}^4 for the PRESENT S-box [20]. We discovered that the only classes reachable by the construction $G(G(\cdot))$ are C_{150}^4 , C_{151}^4 , C_{168}^4 , C_{214}^4 , C_{215}^4 , C_{233}^4 , C_{234}^4 , C_{236}^4 , C_{238}^4 , C_{243}^4 , C_{244}^4 , C_{259}^4 , C_{260}^4 , C_{264}^4 , C_{266}^4 , C_{286}^4 , C_{292}^4 , C_{296}^4 , C_{297}^4 from Table 12 in [20].

[80] states that there are around 2^{26} quadratic 4-bit permutations and they also propose a weighting function to estimate their hardware implementation costs. We used a computer program to select those quadratic permutations, which have smaller hardware implementation costs than the F, G used in [80]. We then checked to which class their composition belongs. and, finally, we selected the one combination of A, B, c, d, G with the smallest hardware implementation. This way we found the following quadratic $G(\cdot)$ such that $S'(\cdot) = G(G(\cdot))$ belongs to class C_{266}^4 :

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
G(x)	0	4	1	5	2	F	B	6	8	C	9	D	E	3	7	A
G(G(x))	0	2	4	F	1	A	D	B	8	E	C	3	7	5	6	9

Let $G(x, y, z, w) = (g_3, g_2, g_1, g_0)$, where x denotes the most significant bit and w the least significant bit of the input, and g_3 denotes the most significant bit and g_0 the least significant bit of the output. Then its *Algebraic Normal Form* (ANF) is as follows: $g_3 = x + yz + yw$, $g_2 = w + xy$, $g_1 = y$ and $g_0 = z + yw$.

Using Definition 8.3.1 we know that the S-box of PRESENT $S(\cdot)$ is linearly equivalent to the found $S'(\cdot) = G(G(\cdot))$, i.e.,

$$S(x) = A(S'(Bx \oplus c) \oplus d) = A(G(G(Bx \oplus c)) \oplus d), \forall x \in \{0, \dots, 15\}.$$

It can be constructed with the following 4×4 -bit matrices A, B and 4-bit constants c, d :

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \quad c = (0001)_2 = 1, \quad d = (0101)_2 = 5.$$

Generally, the quadratic permutation G should be chosen as follows: 1) its $G(G(\cdot))$ must be in the same linearly equivalent class as the given S-box S ; then the minimal hardware implementation for A , B , c , d and G should be selected.

8.3.2 The vertical level

In the second step $G(\cdot)$ has to be divided into three 8×4 vectorial Boolean functions $G_1(\cdot)$, $G_2(\cdot)$ and $G_3(\cdot)$. In practice, all these vectorial Boolean functions are implemented separately. We can reduce the implementation costs by using the following observation:

Observation 8.3.3. The vectorial Boolean functions of $G_1(\cdot)$, $G_2(\cdot)$ and $G_3(\cdot)$ are equal, except for the indices of the inputs and the existence of constants.

The observation is derived from the construction of the vectorial Boolean functions $G_1(\cdot)$, $G_2(\cdot)$ and $G_3(\cdot)$. Again, the left most bit of G_i represents the most significant bit and the right most bit represents the least significant bit, respectively. If we take the latter constructed $G(\cdot)$, then:

$$G_1(x_2, y_2, z_2, w_2, x_3, y_3, z_3, w_3) = (g_{13}, g_{12}, g_{11}, g_{10})$$

$$g_{13} = x_2 + y_2 z_2 + y_2 z_3 + y_3 z_2 + y_2 w_2 + y_2 w_3 + y_3 w_2$$

$$g_{12} = w_2 + x_2 y_2 + x_2 y_3 + x_3 y_2$$

$$g_{11} = y_2$$

$$g_{10} = z_2 + y_2 w_2 + y_2 w_3 + y_3 w_2$$

$$G_2(x_1, y_1, z_1, w_1, x_3, y_3, z_3, w_3) = (g_{23}, g_{22}, g_{21}, g_{20})$$

$$g_{23} = x_3 + y_3 z_3 + y_1 z_3 + y_3 z_1 + y_3 w_3 + y_1 w_3 + y_3 w_1$$

$$g_{22} = w_3 + x_3 y_3 + x_1 y_3 + x_3 y_1$$

$$g_{21} = y_3$$

$$g_{20} = z_3 + y_3 w_3 + y_1 w_3 + y_3 w_1$$

$$G_3(x_1, y_1, z_1, w_1, x_2, y_2, z_2, w_2) = (g_{33}, g_{32}, g_{31}, g_{30})$$

$$g_{33} = x_1 + y_1 z_1 + y_1 z_2 + y_2 z_1 + y_1 w_1 + y_1 w_2 + y_2 w_1$$

$$g_{32} = w_1 + x_1 y_1 + x_1 y_2 + x_2 y_1$$

$$g_{31} = y_1$$

$$g_{30} = z_1 + y_1 w_1 + y_1 w_2 + y_2 w_1$$

Therefore, we only need to implement $G_1(\cdot)$ and then reuse it for $G_2(\cdot)$ and $G_3(\cdot)$ by re-arranging the inputs accordingly. Please note that this observation can also be applied to ANFs containing constants, e.g., the ANFs of the G function specified in the appendix of [80]. In those cases, it is easy to see, that constants can be added to those two shares which initially did not contain a constant to yield three equal vectorial Boolean functions.

Note 8.3.4. $G(\cdot)$ is a 4-bit permutation and its 3-shared version, which is the 12-bit vector $(g_{10}, g_{11}, g_{12}, g_{13}, g_{20}, g_{21}, g_{22}, g_{23}, g_{30}, g_{31}, g_{32}, g_{33})$, is a 12-bit permutation. Therefore, the uniformity property is satisfied.

Assume that the given function $G(\cdot)$ is protected by using a k -share TI and $G_1(\cdot), \dots, G_k(\cdot)$ are its shared functions. Based on the same observation given above, the hardware implementation of $G_i(\cdot), 1 \leq i \leq k$, can be reused. It should be noted

that the shared version of $G(\cdot) = (G_1(\cdot), \dots, G_k(\cdot))$ may not satisfy the uniformity property, e.g., $(G_1(\cdot), G_2(\cdot), G_3(\cdot)) = (g_{10}, g_{11}, g_{12}, g_{13}, g_{20}, g_{21}, g_{22}, g_{23}, g_{30}, g_{31}, g_{32}, g_{33})$ may not be a 12-bit permutation. In this case, we have to use the re-masking method [74] to fulfill this property. Therefore, our technique can be applied not only for this special case but also in general whenever a function is shared.

8.4 Hardware implementation

In this section we will describe hardware implementations of PRESENT protected with the TI countermeasure with a shared data path and an unshared key schedule that is similar to profile 2 described in [80]. This profile is chosen because our approach allows dramatic area savings –up to 57%– within the **S-box** module at the expense of a slight area increase in the storage modules. Thus to allow a fair assessment of the overall area savings, we did not focus on the **S-box** module, but instead implemented a whole PRESENT. Note that we deliberately chose to not share the key schedule for two reasons: first the S-box used would be the same, and second, to focus on the main contribution of this chapter – a new methodology to reduce the area cost of a 4-bit S-box. First we introduce the design flow used before we detail the hardware architectures and finally summarize the implementation results.

8.4.1 Design flow

For the hardware implementation in *VHDL*, we used the Boolean minimization tool **BOOM II** [37, 38] to obtain the four ANFs of G . For functional simulation we used *Mentor Graphics ModelSimXE 6.4b* and *Synopsys DesignCompiler* version *E-2010.12-SP2* was used to synthesize the designs to the *Virtual Silicon* (VST) standard cell library *UMCL18G212T3*, which is based on the *UMC L180 0.18μm*

1P6M logic process and has a typical voltage of 1.8 Volt [47]. For synthesis we advised the compiler to keep the hierarchy and use a clock frequency of 100 KHz.

8.4.2 Serialized architecture

Figure 8.2 depicts our architecture. The main differences between our design and profile 2 in [80] are the **S-box** module and a part of the storage modules for the shared data path. The three shares of the data path are stored in three identical replications of the storage module denoted by **State**, $\mathbf{m}_{d1}$ and $\mathbf{m}_{d2}$. Each of them comprises of 60 flip-flops that can act as a normal 60-bit wide register (vertical shifting direction) or as a 4-bit wide 15 stages shift register (horizontal). The remaining 4 bits are stored in a similar way (denoted with I, II and III in Fig. 8.2) but with two additional 2-to-1 input MUXes (one for each shifting direction). Those 4 bits act as a shift register in a vertical way, allowing to change the input to G . The parallel 60-bit wide output is concatenated with the output of the 4-bit wide register and is transformed by the P-layer of PRESENT. The **Key** module stores the key state and performs the PRESENT key schedule.

The **S-box** module comprises of only *one* 8×4 vectorial Boolean function G (47 GE) that is used for all three shares and for both stages instead of *six* as used in [80]. Recall, that we implement the PRESENT S-box as $S(x) = A(G(G(Bx \oplus c)) \oplus d)$. Therefore, the inputs to G (II and III) are transformed by $Bx+c$ (two times 7 GE) and its output is temporarily stored for two clock cycles in two consecutive 4-bit flip-flops (48 GE). In order to process all shares re-using the same combinational logic, I, II and III are rotated vertically. Suppose the content of I, II and III are denoted a , b , and c , respectively, and the content of the 4 MSB of **Key** is denoted k . Then in the first clock cycle (b, c) are processed yielding t_1 , in the second $(a \oplus k, b)$ yielding t_2 , and finally $(c, a \oplus k)$ yielding t_3 . Since, for the second stage, we do not

Fig. 8.2: Architecture of a serialized TI-PRESENT-80 using our new optimization techniques.

need to process the input to G by $Bx+c$, we transform all three intermediate results t_1, t_2, t_3 by $B^{-1}(x+c)$ (21 GE)¹ and store the results G_1, G_2, G_3 in I, II and III. The aforementioned procedure –computing t_1, t_2, t_3 by rotating I, II and III– repeats for another three clock cycles, after which the second stage is completed. This time the three intermediate results t_1, t_2, t_3 are transformed by $Ax+d$ (18 GE) and stored in the shift registers **State**, m_{d1} and m_{d2} , which are shifting horizontally, and the next 4-bit nibbles are ready to be processed.

The **FSM** module comprises of one initial state, six states for the S-box, one state for the permutation layer that is used instead of the sixth S-box state at the end of each round, a finished state that sets the done signal to high, and a done state. The output is gated by an AND-gate that only lets data pass to the final output XOR after 31 rounds have been processed.

It takes in total $6 * 16 = 96$ clock cycles for one round, hence the output is ready after 2976 clock cycles. During the 16 clock cycles required to output the result

¹ Compared to using two MUXes (19 GE), this approach has the advantage of a simpler control logic at roughly the same area requirements.

nibble-wise, the next message and key can be loaded, which takes 20 clock cycles. Thus in total our architecture requires 2996 clock cycles to process one message, compared to 578 clock cycles reported in [80], which is an overhead of a factor of around 5.2.

8.4.3 Round-based architecture

Based on the serialized implementation we estimate figures for round-based TI PRESENT-80 implementations using the normal TI S-boxes as used in [80] and using our new approach (see Table 8.1). The top-level glue logic consists mostly of XOR and AND gates that scale linearly with the data path width, hence we multiplied it by 16. All storage modules (**Key**, **State**, $\mathbf{m}_{d1}$, $\mathbf{m}_{d2}$) save around 9 GE for a 4-bit MUX that is only required for serialized implementations. The FSM now only consists of a round counter and some simple combinational logic, which we estimate to be around 50 GE. Our approach requires slightly more complex control logic, so we add 50% to be on the safe side. Compared to the normal TI implementation we require two additional 64-bit MUXes (298 GE) for each of the three shares, to allow the shares to be horizontally rotated. The major area increment occurs in the **S-boxes** module, which consists of 16 S-boxes. Due to the pipelining stage in its **S-box** module, the normal TI based implementation requires 2 clock cycles per round, so in total $2 * 31 = 62$ clock cycles. Our approach would require in total $6 * 31 = 186$ clock cycles, which is a timing overhead of only a factor 3.

8.4.4 Discussion

Our main goal is to investigate the savings that one can achieve using our new optimization technique, hence we compare our core to profile 2 as published in [80]. However, there the authors use a combination of clock-gating and scan-flip-flops (s-

8. Improvements in Threshold Countermeasure

132

<i>Serialized Architecture</i>									
	Arch.	Etc.	Key	FSM	State	m_{d1}	m_{d2}	S-box	Sum
D-FF+en	[80]	58	778	139	587	587	587	351	3087
	<i>this work</i>	58	778	146	608	608	608	151	2957
	<i>Difference</i>	0	0	+7	+21	+21	+21	-200	-130
s-FF+cg	[80]	58	520	139	389	389	389	351	2235
	<i>this work*</i>	58	520	146	410	410	410	151	2105
	<i>Difference</i>	0	0	+7	+21	+21	+21	-200	-130
<i>Round-based Architecture</i>									
	Arch.	Etc.	Key	FSM	State	m_{d1}	m_{d2}	S-box	Sum
D-FF+en	<i>based on [80]*</i>	928	769	50	576	576	576	5616	9091
	<i>this work*</i>	928	769	75	874	874	874	2416	6810
	<i>Difference</i>	0	0	+25	+298	+298	+298	-3200	-2281
s-FF+cg	<i>based on [80]*</i>	928	511	50	389	389	389	351	2235
	<i>this work*</i>	928	511	75	687	687	687	2416	5991
	<i>Difference</i>	0	0	+25	+298	+298	+298	-200	-2281

Tab. 8.1: Breakdown comparison of the post-synthesis implementation results of a serialized PRESENT-80 are shown in the upper half and estimates for a round-based implementation are shown in the lower half. For each architecture two different memory strategies are shown: D-flip-flops with enable (D-FF + en), and scan-flip-flops with clock gating (s-FF + cg). All figures are Gate Equivalents (GE) and estimated figures are indicated with an *.

FF+cg), which results in storing costs of 6 GE per bit (plus a negligible overhead for clock gating logic). For ASIC prototyping it is sometimes not desirable to use clock gating, thus we decided to use D-flip-flops with enable signal (D-FF+en), which results in storage costs of 9 GE per bit.

In order to have a fairer comparison between our results and [80], we also report post-synthesis figures for a modified variant of their source code where we replaced the clock gating and scan-flip-flops with D-flip-flops with enable (9 GE) (see Table 8.1) We have also estimated the area requirements of our implementation using 6 GE scan-flip-flops in combination with clock gating.

Please note that the area of 387 GE for the *S-box* module in [80] comprises of both the shared S-box (359 GE) for the data path and the unshared S-box (28 GE)

for the *Key schedule*. Thanks to a more optimized representation [41], the unshared PRESENT S-box we used only takes 22 GE, and since the unshared S-box is only used in the *Key Schedule* module we account its area share there. Though our post-synthesis results are smaller for the S-box, FSM, and top level glue logic (etc.), we have taken the figures from [80] to ensure a fair comparison.

As one can see, for the serialized architecture, the top level glue logic and the **Key** module are identical, while the control logic (**FSM**) is slightly more complex for our approach. Compared to [80] our approach requires six additional 4-bit wide 2-to-1 MUXes, which increase the area requirements of the storage components by 21 GE each. In total it is possible to save 130 GE, or around 37-40% of the area of the **S-box** by using our new approach. The overall savings are determined by the area share of the **S-box** module, which is dependent on the implementation strategy. For the serialized implementation described above, due to the minor contribution of the **S-box** module to the overall area requirements of only around 11-16%, in the end we achieve savings of only around 4-6%, while our estimates for a round-based implementation, where the area share of the **S-box** module is 62-68%, indicate overall savings of 25-27%. Table 8.2 summarizes these facts, and highlights that our new approach is more rewarding for architectures with a larger area share of the **S-box** module.

Architecture Strategy	S-box Savings	Storage Strategy	S-box Area share	Overall Savings
serial	-37.0%	D-FF + en	11.4%	-4.2%
		s-FF + cg	15.7%	-5.8%
round-based	-40.6%	D-FF + en	61.8%	-25.1%
		s-FF + cg	67.9%	-27.6%

Tab. 8.2: Area savings for different implementation strategies.

8.5 Experimental results

We analyzed power consumption traces obtained from a SASEBO G-II in order to evaluate the security of our new design. First, the measurement setup is introduced and a classical DPA analysis is performed to compare the results with [80] and prove that our implementation offers the same security level. Second, we demonstrate that the implementation of [80] is vulnerable against Mutual Information Analysis (MIA) in practice (as theoretically shown in [74]) and compare the strength of this attack with the correlation-enhanced collision attack targeting higher order moments as described in [69]. Last, we shortly revisit Wagner's attack in the context of three-share Threshold Implementations.

8.5.1 Measurement setup

The SASEBO G-II hosts two FPGAs, i.e., a control FPGA (Xilinx XC3S400A-4FTG256, Spartan-3A series) and a cryptographic FPGA (Xilinx XC5VLX50-1FFG324, Virtex-5 series) which is decoupled from the rest of the board in order to minimize electronic noise from the surrounding components. It is supplied with a voltage of 1V by an external stabilized power supply as well as with a 3MHz clock (from an on-board clock oscillator). The power consumption is measured over a 1Ω resistor inserted in the VDD line by using a differential probe. All power traces are collected with a LeCroy WR610Zi-s-32 oscilloscope at a sampling rate of 1GS/s.

8.5.2 Side-channel resistance

We will now show that our implementation, which was presented in the previous section, has the same security level against the classical attacks as the implementation described in [80].

Fig. 8.3: Exemplary power trace of serialized implementation.

Figure 8.3 shows an exemplary power trace of the first round of an encryption run as well as a zoomed extract. The high peaks in the power consumption on the left side of Figure 8.3(a) are caused by the loading procedure of the plaintext and key to the cryptographic FPGA. The encryption starts at sample 8500 - for our analyses we omit these first 8500 samples. In Figure 8.3(b) one can clearly identify the peaks in the power consumption for every single clock cycle (333 samples between the peaks equal 3 MHz).

We conducted our evaluation based on 5,000,000 power traces. As in [80] we considered three different attack models for the DPA attack: *Hamming Weight* (HW) of one nibble of the S-box input, HW of one nibble of the S-box output and the *Hamming Distance* (HD) between two nibbles of consecutive S-box states. Since all attacks targeted 4 bits, 16 key guesses had to be analyzed. The results of the DPA attack for the three models are depicted in Figure 8.4. One can observe from the plots –as intuitively expected– that none of the attack models reveals the correct key hypothesis. Hence we can conclude that our implementation has no first order leakage.

In [80] the authors chose the HD of two consecutive state nibbles as the most promising leakage model. We found that attacking the HD between two consecutive

(a) HW of the S-box output (b) HD of subsequent state nibbles

(c) HW of S-box input

Fig. 8.4: DPA on sequential TI countermeasure implementation.

nibbles of the G -stage output (see Figure 8.1) shows higher leakage. Figure 8.5 shows exemplary correlation results (using known masks) comparing said models. When using the HD of two consecutive state nibbles as the model, one can nicely determine the repeating peaks caused by the shift to update the state, but the correlation coefficient is approximately five times lower than for the attack using our proposed model. We can only assume that this behavior is caused by a higher load of the output flip-flops of the G -stage, since the number of bits attacked and the number of bits which flip at the attacked point in time are the same for both. Hence, we chose the output of the G -stage as the target for our subsequent attacks. In addition, Figure 8.5 shows how many traces are needed until the correct key hypothesis is revealed. The original model needs approximately 2000 where as our proposed model needs only 500 traces.

(a) HD of subsequent S-box states (b) HD of subsequent intermediate G states

(c) Number of traces at sample 1606 (d) Number of traces at sample 607

Fig. 8.5: DPA results analysing 200,000 traces comparing two models, over number of traces (with known masks).

As already mentioned we now want to investigate the resistance of the TI countermeasure against more sophisticated attacks. In [74] the authors showed, based on simulations, that it is possible to attack a Noekeon S-box secured by TI with MIA when all shares are processed in parallel. Unfortunately, a practical evaluation is still missing. Therefore, we can not say how efficient the attack is or if it works at all in practice, i.e., in the presence of noise. In [69] the author mounted a correlation-enhanced collision attack on the parallel implementation described in [80] and demonstrated that it can be practically broken targeting higher-order moments. We measured 10,000,000 traces of this implementation and mounted both attacks targeting the HD of consecutive outputs of the G -stage as described above (see Figure 8.6).

Fig. 8.6: Advanced attacks on parallel implementation of the TI countermeasure.

To the best of our knowledge this is the first successful practical attack of MIA targeting the TI countermeasure (see Figure 8.6(a)). Roughly 9,000,000 measurements are required until the correct key nibble is revealed. Figure 8.6(b) shows the result of a correlation-enhanced collision attack targeting second-order moments. With this attack the correct key nibble is already revealed after approximately 6,000,000 measurements which proves that this attack is more powerful than MIA in this scenario. We repeated both attacks against our implementation. Figure 8.7

depicts the results of both attacks, none of them succeeded. This is not a surprising result as our proposed model does not fit the implementation anymore. It should be noted though that while our S-box is serialized, there are still sources for univariate leakage in our implementation, e.g. the state update, which might be exploited with more appropriate models. To secure the implementation against these kind of attacks the whole implementation must be carefully serialized for every clock cycle, which is ongoing work.

Fig. 8.7: Advanced attacks on serialized implementation of the TI countermeasure.

Last, we want to revisit Wagner’s zero-offset attack described in [92]. The paper shows that it is theoretically possible to attack implementations where masks and masked values are processed in parallel, as it is the case for the parallel implementation of [80] which was attacked above. We performed the zero-offset attack against said implementation without success. Subsequently, we performed simulations of the zero-offset attack against TI with three shares and a classical masking scheme with two shares. As it turned out, the attack works against the two-share masking scheme (albeit it is very sensitive to noise) but it does not work against masking schemes utilizing three shares like TI. To be able to attack three shares one has to modify the attack, i.e., by raising the mean-free measurement values to the power of three instead of squaring. However, for the attack to work we had to analyze 100

times more simulated power traces than for a MIA, which shows how sensitive this attack is against noise and why it does not work in practice.

8.6 Conclusion

In this chapter we introduced two methodologies to efficiently implement 3-share TI to a given S-box. Applying these methodologies to the PRESENT S-box we were able to reduce its area requirement by 37-40% (130 GE), resulting in the smallest implementation of a protected PRESENT known in literature so far (2105 GE). Due to the small area share of the S-box in a serialized implementation, the overall savings were only around 4-6% at a timing overhead of a factor of 5.2. However, different implementation strategies, like round-based architectures, promise greater savings of more than 25% while having a smaller timing overhead of only 3.

We have practically shown that our approach achieves a similar security level as previously introduced implementations. In addition, we presented the first practical MIA against a parallel implementation of the TI and demonstrated that it is inferior to a correlation-enhanced collision attack using second-order moments.

It is also noteworthy to point out that our contribution allows to reduce the memory requirements of software implementations of S-boxes protected by the TI countermeasure by a factor of six. Future work will include investigations in this direction.

9. ENABLING 3-SHARE THRESHOLD COUNTERMEASURE TO ANY 4-BIT S-BOX

Threshold Implementation (TI) is an elegant and widely accepted countermeasure against 1-st order Differential Power Analysis (DPA) in Side Channel Attacks (SAC). The 3-share TI is the most efficient version of TI, but so far, it can only be applied to 50% of all 4-bit S-boxes. In this chapter, we study the limitations of decomposition and introduce factorization to enable the 3-share TI to any 4-bit S-box. To highlight the strength of our new methodology, we construct 3-share TIs of S-boxes of SERPENT and 3-share TIs of several 4-bit S-boxes which were believed to be not possible until now.

This chapter is based on the joint work with Sebastian Kutzner, Prof Axel Poschmann. The factorization idea is derived and completed through discussion with Sebastian Kutzner and Prof Axel Poschmann. Thank you Prof Axel Poschmann and Sebastian Kutzner for editing this chapter.

9.1 Introduction

The number of shares required for a Threshold Implementation depends on the degree d of the non-linear function (S-box) and [73–75] have shown that it is at least $d + 1$. It implies that the higher the degree of the non-linear function, the more shares are required and the larger is the implementation. Since a degree of two is

the minimal degree of a non-linear function, the optimal number of shares is three. Therefore, to apply a 3-share Threshold Implementation to a larger degree function, this function must be represented as a composition of quadratic functions [80]. We call this method *sequential structure* due to its format. According to [74,80] we have two stages in applying 3-share TI. In first stage, we decompose a given S-box into quadratic permutations and we call this stage *Decomposition Stage*. In next stage, we make 3-share TIs of all those quadratic permutations being 12-bit permutations and we call this stage *Sharing Stage*.

According to [20] all 4-bit S-boxes which can be protected by 3-share TI based on sequential structure belong to the alternating group A_{16} of the symmetric group S_{16} . This answer implies that we can not apply 3-share TI to those S-boxes which do not belong to A_{16} , i.e., $B_{16}=S_{16} \setminus A_{16}$. Therefore, we introduce the *factorization structure* which is extended structure of sequential structure. This new idea has a big contribution: it allows us to decompose any 4-bit S-box to quadratic vectorial boolean functions and hence, enabling us to apply 3-share TI to *any* given 4-bit S-box. To support our claims we show how to apply 3-share TI to SERPENT and 3-share TIs of some different 4-bit S-boxes, what, up to now, were believed to be not possible.

The chapter is organized as follows. In Section 9.2, we recalled the result of 3-share TI of PRESENT S-box. Then we will discuss how to decompose any 4-bit S-box into quadratic decompositions, i.e., studying Decomposition Stage in Section 9.3. After that, we show how to make 3-share TI of each quadratic decomposition being a 12-bit permutation in Sharing Stage in Section 9.4. In Section 9.5, we presented 3-share TIs for S-boxes and important permutation in B_{16} which are believed impossible to apply 3-share TI. Section 9.6 concludes our chapter.

9.2 3-share TIs to cubic 4-bit S-boxes

In this section we revisit the results of [80] describing a 3-share TI of PRESENT. Since the PRESENT S-box $S(\cdot)$ is a cubic 4-bit permutation, the minimum number of shares is 4 [74]. To apply 3-share TI, the S-box is decomposed into two quadratic permutations $S(\cdot) = F(G(\cdot))$ as shown in Figure 9.1. We call this method **sequential structure**.

Fig. 9.1: Composition of the PRESENT S-box [80].

According to [74, 80] there are two stages in applying 3-share TI to a 4-bit S-box when using a sequential structure:

1. **Decomposition:** Finding the decompositions of a given S-box, which are required to be quadratic permutations.
2. **Sharing:** Constructing 3-share TIs for those quadratic permutations. Their 3-share TIs should fulfill Uniformity, i.e., their 3-share TIs are 12-bit permutations.

Note 9.2.1. In the Sharing stage, it is easy to do 3-share TI to any 4-bit at most quadratic permutation. Unfortunately it does not guarantee that its 3-share TI is a 12-bit permutation, i.e., it is not guaranteed that the Uniformity property is satisfied. For that purpose, the so-called *remasking* technique [74] may be applied, which remasks the input(s) of next round or the input(s) of the next function with fresh (and uniformly distributed) random bits.

In this chapter, we discuss how to decompose an arbitrary 4-bit S-box first (Decomposition stage), before we show how to obtain its 3-share TIs of decompositions which are 12-bit permutations. It means that 3-share TIs satisfy Uniformity without using the remasking method, which is a significant advantage, as the generation of random bits suitable for cryptographic or masking purposes can be very expensive on embedded devices.

Note 9.2.2. In order to apply 3-share TI to any arbitrary 4-bit S-box, we have to extend the Decomposition and the Sharing stages. Those extensions yield the **factorization structure**, our main contribution in this chapter. We will detail the extensions required in Section 9.3.2.

9.3 Decomposition Stage of Threshold Implementation

In this section we investigate the decomposability of 4-bit S-boxes. In Section 9.3.1, we will recall the results from [20]. If a 4-bit S-box is decomposed in a sequential structure then it must be in the alternating group A_{16} . In order to apply 3-share TI to the remaining 50% of 4-bit S-boxes which are in B_{16} , we extend the sequential structure by using our new idea: **Factorization**, which yields the **factorization structure**. Our contribution allows to decompose any 4-bit S-box.

9.3.1 Decomposition of 4-bit S-boxes using a sequential structure

Definition 9.3.1. If a vectorial Boolean function $S(\cdot)$ can be written as a composition of several lower degree vectorial Boolean functions $H(\cdot), G(\cdot), \dots, F(\cdot)$, i.e., $S(\cdot) = F(\dots G(H(\cdot)))$ then $H(\cdot), G(\cdot), \dots, F(\cdot)$ are called the decompositions of $S(\cdot)$.

If S is a permutation, then all its decompositions $H, G, \dots, F$ have to be

permutations as well. Hence, if S is a 4-bit permutation then $H, G, \dots, F$ are also 4-bit permutations. We recall some properties of a permutation in S_{16} .

Lemma 9.3.2. [48] A_{16} is a subgroup of S_{16} , i.e if $H(\cdot)$ and $G(\cdot)$ are permutations in A_{16} then its composition permutation $F(\cdot) = G(H(\cdot))$ or $F(\cdot) = H(G(\cdot))$ must be in A_{16} as well.

Lemma 9.3.3. [20] All linear and quadratic permutations in S_{16} are in A_{16} .

Theorem 9.3.4. [20] If a permutation $F(\cdot)$ is able to be represented as a composition of quadratic permutations, then $F(\cdot)$ is in A_{16} .

However, 50% of all 4-bit S-boxes are not decomposable using a sequential structure, i.e., all those S-boxes belong to B_{16} . Hence, there is no method known so far on how to apply 3-share TIs to those S-boxes. We now introduce a new methodology to solve this open problem.

9.3.2 Decomposition of 4-bit S-boxes using a factorization structure

We start with a very simple example, i.e., decomposing a cubic term. The *Algebraic Normal Form* (ANF) of a cubic 4-bit S-box contains at least one cubic term. Without loss of generality, we assume that in the cubic part there is only one cubic term $T(w, z, y, x) = (d, c, b, a) = (xyz, 0, 0, 0)$. The input bits of T are x, y, z, w and the output bits of T are a, b, c, d . The left most bit represents the most significant bit and the right most bit represents the least significant bit, respectively. The ANF of T is:

$$d = xyz,$$

$$c = 0,$$

$$b = 0,$$

$$a = 0.$$

We also can write T as follows:

$$T(\cdot) = F(G(\cdot)) \oplus V(\cdot),$$

where F , G and V are the following quadratic vectorial Boolean functions:

$$\begin{array}{lll}
 & d = xy \oplus w, & d = zw, & d = zw, \\
 G : & c = z, & F : & c = 0, & V : & c = 0, \\
 & b = 0, & & b = 0, & & b = 0, \\
 & a = 0. & & a = 0. & & a = 0.
 \end{array}$$

As one can see, using this approach, it is possible to represent a cubic term of an ANF by a set of quadratic vectorial Boolean functions. By applying this approach term-by-term, it is possible to decompose any cubic vectorial Boolean function, *including odd permutations*. In order to make our idea clear, we provide as another example, the 4-bit odd permutation $M=[0, 1, 2, 3, 4, 5, 6, 15, 8, 9, 10, 11, 12, 13,$

14, 7]. Its ANF is:

$$d = w \oplus xyz,$$

$$c = z,$$

$$b = y,$$

$$a = x.$$

Please note that the example above is only *one* out of many choices for M . We chose M for its simplicity and implementation efficiency. The permutation M can be factorized as follows:

$$M(\cdot) = M_2(M_1(\cdot)) \oplus M_3(\cdot).$$

Where the ANFs of M_1 , M_2 , and M_3 are:

$$\begin{array}{lll} M_1 : & \begin{array}{l} d = yx \oplus w, \\ c = z, \\ b = y, \\ a = x. \end{array} & \begin{array}{l} M_2 : \\ \\ \\ \end{array} \begin{array}{l} d = zw, \\ c = 0, \\ b = 0, \\ a = 0. \end{array} & \begin{array}{l} M_3 : \\ \\ \\ \end{array} \begin{array}{l} d = zw \oplus w, \\ c = z, \\ b = y, \\ a = x. \end{array} \end{array}$$

For any given 4-bit S-box S we always can find a set of quadratic vectorial Boolean functions F_i , G_i , $1 \leq i \leq n$, and V such that:

$$S(\cdot) = \bigoplus_{i=1}^n F_i(G_i(\cdot)) \oplus V(\cdot).$$

We call the format above **factorization structure** and we summarize the idea

for the Decomposition stage in Figure 9.2.

Fig. 9.2: Flowchart of the Decomposition stage for a given 4-bit S-box.

9.3.3 Decomposition of 4-bit S-boxes using a hybrid structure

We will discuss the Sharing stage in Section 9.4 where we ensure that the 3-share TI of a 4-bit S-box is a 12-bit permutation, i.e., the 3-share TI fulfills Uniformity. Please note that the Sharing stage for sequential structures is not complicated, and we will show how to solve this problem in Section 9.4.1.

Sharing a factorization structure, however, without using the remasking method to fulfill the Uniformity is a challenge, and it is extremely difficult for 4-bit S-boxes which have many cubic terms. In order to make the workload in the Sharing stage easy, we propose the following approach:

1. If a given S-box S is in A_{16} (S is an even permutation) then we use the result in Section 9.4.1.

2. If a given S-box S is in B_{16} (S is an odd permutation) then:
 - (a) Construct a 4-bit odd permutation M that can be shared into a 12-bit permutation, for example $M=[0, 1, 2, 3, 4, 5, 6, 15, 8, 9, 10, 11, 12, 13, 14, 7]$. This permutation is decomposed by using a factorization structure and its 3-share TI can be made being a 12-bit permutation, i.e., permutation M satisfies Uniformity. This will be shown in Section 9.4.2.
 - (b) Since M and S are odd permutations, the permutation S' such that $S(\cdot) = M(S'(\cdot))$ is an even permutation, i.e., S' is in A_{16} [48]. Then we apply the result in Section 9.4.1 to share S' .

Actually, the method above is a **hybrid structure** between a sequential structure and a factorization structure. This hybrid structure is very useful, because it can help us to fulfill the uniformity property without using remasking, thus we will work with the hybrid structure instead of a plain factorization structure. Figure 9.3 depicts the hybrid structure.

Fig. 9.3: Hybrid Structure.

So far, we already present how to decompose a given 4-bit S-box. If the given S-box is in A_{16} then we use a sequential structure, otherwise we use a hybrid structure which is a mixture of a sequential structure and a factorization structure. The Decomposition Stage is done, so we can move on to the Sharing stage, which will

be treated in the next Section. It is interesting to see how to make a 3-share TI of those decompositions being 12-bit permutations, i.e., the Uniformity is fulfilled, without using remasking method. Note that the degrees of all decompositions is 2, hence, we can construct the 3-share TIs.

9.4 Sharing stage of Threshold Implementation

In this section, we discuss how to make 3-share TIs of decompositions of a given 4-bit S-box being 12-bit permutations. Since the given S-box can belong to A_{16} or B_{16} , we have two structures: a sequential structure, which is treated in Section 9.4.1, and a hybrid structure, which is composed of a sequential structure and a factorization structure. Consequently, in Section 9.4.2 we will treat the factorization structure, and we will also discuss its security, i.e., why it resists first order DPA in the presence of glitches and why it satisfies uniformity.

9.4.1 Sharing stage using a sequential structure

In this subsection, we discuss how to share the decompositions of a given S-box S in A_{16} . We adopt the sharing in [73, 74, 80]. For a given function $F(w, z, y, x)$ and inputs x, y, z, w , they are split into 3 shares $F_1, F_2, F_3, x_1, \dots, w_3$. For monomials involving two indices, it is obvious which F_i to place them in. For example, we must place monomials y_1w_2 and z_2w_1 in F_3 . For monomials involving just one index, e.g., x_1 or y_2w_2 , we adopt the convention that terms with index 1 (resp. 2, 3) are placed in F_3 (resp. F_1, F_2). The constant term is placed in F_1 . In [20], this approach is called *direct sharing*. According to [20], all 4-bit permutations in A_{16} can be decomposed by using a sequential structure and their 3-share TIs satisfy Uniformity by simply using direct sharing.

9.4.2 Sharing stage using a factorization structure

The Sharing stage for S-boxes in B_{16} is detailed here. As we have already pointed out, it is better to use a hybrid structure to decompose an S-box in B_{16} . In the previous section, based on [20], we already saw that all 3-share TIs of 4-bit permutations in A_{16} can be made being 12-bit permutations by using direct sharing. Hence, we only focus on 3-share TI of M , i.e., the 3-share TIs of the quadratic vectorial Boolean functions M_1, M_2, M_3 . Let denote $M_s, M_{s_1}, M_{s_2}, M_{s_3}$ as 3-share TIs of M, M_1, M_2, M_3 , respectively.

The ANF of 12-bit M_{s_1} of M_1 is:

$$d_1 = w_2 \oplus y_2x_2 \oplus y_2x_3 \oplus y_3x_2,$$

$$d_2 = w_3 \oplus y_3x_3 \oplus y_1x_3 \oplus y_3x_1,$$

$$d_3 = w_1 \oplus y_1x_1 \oplus y_1x_2 \oplus y_2x_1,$$

$$c_1 = z_2,$$

$$c_2 = z_3,$$

$$c_3 = z_1,$$

$$b_1 = y_2,$$

$$b_2 = y_3,$$

$$b_3 = y_1,$$

$$a_1 = x_2,$$

$$a_2 = x_3,$$

$$a_3 = x_1.$$

The ANF of 12-bit M_{s_2} of M_2 is:

$$d_1 = z_2w_2 \oplus z_2w_3 \oplus z_3w_2,$$

$$d_2 = z_3w_3 \oplus z_1w_3 \oplus z_3w_1,$$

$$d_3 = z_1w_1 \oplus z_1w_2 \oplus z_2w_1,$$

$$c_1 = 0,$$

$$c_2 = 0,$$

$$c_3 = 0,$$

$$b_1 = 0,$$

$$b_2 = 0,$$

$$b_3 = 0,$$

$$a_1 = 0,$$

$$a_2 = 0,$$

$$a_3 = 0.$$

The ANF of 12-bit M_{s_3} of M_3 is:

$$d_1 = w_2 \oplus z_3 w_3 \oplus z_2 w_3 \oplus z_3 w_2,$$

$$d_2 = w_3 \oplus z_1 w_1 \oplus z_1 w_3 \oplus z_3 w_1,$$

$$d_3 = w_1 \oplus z_2 w_2 \oplus z_1 w_2 \oplus z_2 w_1,$$

$$c_1 = z_2,$$

$$c_2 = z_3,$$

$$c_3 = z_1,$$

$$b_1 = y_2,$$

$$b_2 = y_3,$$

$$b_3 = y_1,$$

$$a_1 = x_2,$$

$$a_2 = x_3,$$

$$a_3 = x_1.$$

Then $M_s = M_{s_2}(M_{s_1}(\cdot)) \oplus M_{s_3}(\cdot)$ is a 12-bit permutation. In other words, the 3-share TI of M fulfills uniformity.

Note 9.4.1. Studying Figure 9.3 allows to observe the following:

1. Among three quadratic vectorial Boolean functions M_1 , M_2 , M_3 , only M_1 is a 4-bit permutation and M_{s_1} is a 12-bit permutation. Hence, M_1 satisfies the uniformity property.
2. It is obvious that M_{s_2} and M_{s_3} do not fulfill the uniformity property. However, both functions are *not required to satisfy uniformity*, only their XOR sum has to (as this will potentially be the input to a subsequently shared function).

Instead both functions M_{s_2} and M_{s_3} only need to satisfy non-completeness and correctness, and their inputs need to fulfill Equation 8.1, in order to resist first order DPA in the presence of glitches (Theorems 2 and 3 in [73–75]).

3. The output of M_s is the result of XORing the outputs of M_{s_2} and M_{s_3} . Since an XOR is a linear operation (i.e. having degree 1), only outputs from the same share are combined together. This means, this operation is first order DPA resistant in the presence of glitches, as potential leakage will only depend on a single share, but information of all three shares are required for a successful DPA.
4. Since M is in B_{16} and M_s is a 12-bit permutation, the 3-share TI of M satisfies uniformity. It means our proposed hybrid structure is secure.

Until now, we already have shown how to successfully decompose and share an arbitrary 4-bit S-box. To do that, we have studied and developed the sequential structure and the hybrid structure (or an improved version of factorization structure).

9.5 3-share TIs of S-boxes in B_{16}

In this section, we list decompositions of S-boxes in B_{16} , that are used by the following algorithms: CLEFIA [31], DES [71], DESL [57], GOST [96], HUMMINGBIRD1 [36], HUMMINGBIRD2 [35], LUCIFER [87], mCrypton [60], SERPENT [14], TWINE [88], TWOFISH [85], and the Inversion (x^{-1}) function in $GF(2^4)$ which is used in mCrypton [60] or in [70]. Previously, all of these algorithms could not have been protected by TI, but thanks to the hybrid structure it is possible.

All examples use the odd permutation $M=[0, 1, 2, 3, 4, 5, 6, 15, 8, 9, 10, 11, 12,$

13, 14, 7] which is also used in previous sections. Recall, that M can be any odd permutation of which the shared version is a 12-bit permutation, i.e., satisfying the uniformity property without using remasking.

Fig. 9.4: Decompositions of S-boxes.

For the sake of convenience, a given permutation is described in hexadecimal representation. For example, if a permutation $F=[15, 5, 6, 14, 13, 7, 2, 10, 8, 0, 11, 1, 12, 4, 9, b]$, then F is written as follows: $F=f56ed72a80b1c493$. All S-boxes in this section can be found in [20, 84] or from their respective specifications.

All S-boxes below belong to B_{16} and they can be decomposed in two different ways (see Figure 9.4):

- **type 1:** $S(\cdot) = M(F(G(G(\cdot))))$
- **type 2:** $S(\cdot) = M(F(G(\cdot)))$

In fact nearly all S-boxes belong to type 1 and only two S-boxes (iS_4 of HUMMINGBIRD2 and S_5 of SERPENT) belong to type 2. The 3-share TIs of all F and G by using direct sharing are 12-bit permutations.

CLEFIA [31]

1. SS_0 : F=e6ca89d24b10537f, G=021346fda89bce57;
2. SS_1 : F=6f29a5e3781cd4b0, G=053f8db72694ae1c;
3. SS_2 : F=b56e7302da981cf4, G=094c187f2b6e3a5d;
4. SS_3 : F=a6d295c37bf048e1, G=02319b8a57ec46df;

DES[71]

Actually, the i – th DES S-box ($DESi$) contains a set of four 4-bit S-boxes. Notation $DESi_j$ means the j – th row (i.e. 4-bit S-box) of the i – th DES S-box.

1. $DES2_0$: F=f986bda42c710e35, G=4c28a0f7d539b16e;
2. $DES2_1$: F=acd1265b97403fe8, G=1c593a7f0d482b6e;
3. $DES2_2$: F=d3f0c481b596a2e7, G=9d26a78503cf4e1b;
4. $DES2_3$: F=dc8b37421f6a5e09, G=0b1a46579382decf;
5. $DES3_0$: F=d69e75a410f2b3c8, G=168c079d24be35af;
6. $DES3_1$: F=803a46ed952f17bc, G=17069a8b5243dfce;
7. $DES3_2$: F=df47ae50b921c836, G=0d861c972ea53fb4;
8. $DES3_3$: F=9716fac0b8e4d532, G=fb647ec318a59d02;
9. $DES4_0$: F=fd3402cb75168ae9, G=419b03c8de62fa57;
10. $DES4_1$: F=3edf68ba70c41592, G=125ac68e9bd34f07;
11. $DES4_2$: F=abc4fd928375e610, G=094b6a285d1f3e7c;

12. $DES4_3$: $F=36dea581b2f047c9$, $G=02d64f135e8a9bc7$;
13. $DES5_0$: $F=28fc1b569a7d304e$, $G=0e1f869725bcad34$;
14. $DES6_0$: $F=792bd3c54a81e06f$, $G=4e396f18a0d7c5b2$;
15. $DES6_3$: $F=48ac537b2e9f601d$, $G=0a7c1e68295f3d4b$;
16. $DES7_0$: $F=6b3d719c2e5a8f40$, $G=21e74da903c56f8b$;
17. $DES7_1$: $F=68f143bc970ead52$, $G=be364f290c1d57a8$;
18. $DES7_2$: $F=abd4c93e671805f2$, $G=1a084e5c293b7d6f$;
19. $DES8_0$: $F=d572c908143be6af$, $G=0eb4962c1da7853f$;
20. $DES8_1$: $F=fd963b2745c01ae8$, $G=1c0d3a2b59487f6e$;
21. $DES8_2$: $F=fa41e5830b6d72c9$, $G=048c9d152f6b3e7a$;

DESL [57]

1. Row_0 : $F=e6a3d4197f2b5c80$, $G=091d7f6b5c482a3e$;
2. Row_1 : $F=51ebc9378d6204af$, $G=02cf1b5e93d68a47$;
3. Row_2 : $F=15dbef74c2a63809$, $G=17ad358f269c04be$;
4. Row_3 : $F=dae51379f80b64c2$, $G=af53269e8d7104bc$;

GOST [96]

1. k_3 : $F=52840cadb79e613f$, $G=063d1f24acb5978e$;
2. k_4 : $F=f93457dec1a62b08$, $G=0e7d1b4a2c5f3968$;

3. k_7 : F=d7954f6b2c08e1a3, G=0a6f384c1b7e295d;
4. k_8 : F=5b79d3f104ae62c8, G=179fda52e46cb038;

HUMMINGBIRD1 [36]

1. S_0 : F=82f7e639c40ab1d5, G=0f1e9687bd24ac35;
2. S_1 : F=063b7f42d1eca895, G=0f861e97ad24bc35;
3. S_2 : F=21430895dbeca76f, G=0ad7b16c92e54f38;
4. S_3 : F=0f2e7d5c4a6b3819, G=0a7f295c6e1b4d38;

HUMMINGBIRD2 [35]

1. S_1 : F=f56ed72a80b1c493, G=0a5bd38217ce469f;
2. S_2 : F=a8034ce7b61d52f9, G=14860d9fae3cb725;
3. S_3 : F=2f6e5d1c4a380b79, G=0f5bc78293d64a1e;
4. S_4 : F=0819ae37c4d562fb, G=853b29a47ed1f06c;

The inverse S-boxes of HUMMINGBIRD2:

1. iS_1 : F=0d42ca8597eb631f, G=3c4b21de56a9780f;
2. iS_2 : F=de8c94b162305f7a, G=14a69d2fcbe05378;
3. iS_3 : F=c36740b18e5d2fa9, G=0c6f2a583b491d7e;
4. iS_4 : F=f5ac403b16927ed8, G=209a8b3164fced75. (type 2)

Inversion (x^{-1}) in $GF(2^4)$

The function $x^{-1}=019edb76f2c5a438$ which is defined over $GF(2)/(x^4 \oplus x \oplus 1)$.
 $F=843dae67f25bc91$ and $G=059dbf278e3416ac$.

LUCIFER [87]

1. S_0 : $F=a2fde8b7906534c1$, $G=1e482c6b0f593d7a$;
2. S_1 : $F=f21deb047c93658a$, $G=068f9e174bd3c25a$;

mCrypton [60]

1. S_0 : $F=4af0827c3516b9de$, $G=0a7c5e28396d4f1b$;
2. S_1 : $F=19df3b647580cea2$, $G=06d71fce8a5b9342$;
3. S_2 : $F=31078f46ec25ad9b$, $G=2b5f097d4e186c3a$;
4. S_3 : $F=b420af918c7e3d65$, $G=041d3f26ac97b58e$;

SERPENT [14]

1. S_3 : $F=072e351c9db4af86$, $G=0c792f5a3e4b1d68$;
2. S_4 : $F=53bd19f708e6ca24$, $G=ea5d69cf0873214b$;
3. S_5 : $F=7c4b259a3e6f01d8$, $G=05432761c89feabd$. (type 2)
4. S_7 : $F=18679d3f5acb024e$, $G=0d87961c3fa4b52e$;

The inverse S-boxes of S_3 , S_4 , S_5 , S_7 :

1. iS_3 : $F=09dacef3b1624578$, $G=0c483e7a6f2b195d$;
2. iS_4 : $F=98b7406fac5e21d3$, $G=1a0bc2d34e5f8796$;

3. iS_5 : F=87f6dc43b915e2a0, G=0eb63c95842da71f;
4. iS_7 : F=35f921edc60a874b, G=0c489d5173bfa6e2;

TWINE [88]

1. S : F=d2305ebc7a98f614, G=bda5e92c0687431f;

TWOFISH [85]

1. $q1, t1$: F=a0f2d785c139b64e, G=0c483e7a6f2b195d;
2. $q1, t0$: F=2847ba6e1c9d350f, G=069c8d1734aebf25;
3. $q0, t0$: F=50d87b3fa6e29c14, G=b4ace16058732f9d;
4. $q0, t2$: F=456f09ba23e781dc, G=0d841cb73e952fa6;

9.6 Conclusion

The 3-share TI is an efficient and well-accepted countermeasure against DPA. Until now, it was only possible to apply this countermeasure to 4-bit S-boxes which belong to A_{16} (roughly 50 % of all S-boxes). We introduce a new methodology called factorization. This idea enables us to decompose S-boxes not belonging to A_{16} . More importantly though, since all S-boxes are factorized to linear and quadratic vectorial Boolean functions only, we can apply 3-shares TI to all S-boxes, which were believed to be not possible until now.

10. CONCLUSION

In this thesis, we have studied the cryptanalysis and design of several symmetric-key ciphers, i.e, *ZUC* stream cipher, *SERPENT*, *KASUMI*, *SMS4* blockciphers. Additionally, we have developed several methodologies which enhance the Threshold Implementations against first order DPA and enable 3-share TI to all 4-bit S-boxes. As a result of theses findings, we have identified some open problems which can be addressed in the future research.

Open Problem I: Studying the Linear Hull.

The result of (Multiple) Linear Cryptanalysis are strongly based on the computation of bias(es) of linear approximation(s). The experiment on 5-round *SERPENT* in Chapter 5 has shown that the χ^2 distinguisher can work very well for certain keys, i.e, the advantages are very high for some experimental secret keys, but are almost equal to zero for the other ones. The explanation of the experiments is the linear hull effect [76]. The linear hull effect states that depending on the given secret key, the true bias of a considered linear approximation can be much higher (or weaker) than that computed by using the Piling-Up Lemma. Hence, we conclude that more studies in Linear Hull will be the key to answer accurately some computational open problems in (Multiple or Multidimensional) Linear Cryptanalysis.

Open Problem II: Does there exist an extension model of Algorithm 1 of Multidimension Linear Cryptanalysis for Feistel ciphers?

In [49], the linear approximations with the same key mask are used (denoted

as type-1) and type-1 linear approximations only appear in Feistel ciphers. In the Algorithm 1 of MLC, the linear approximations have different key masks and text masks (denoted as type-2). Hence, our open problem is to find if exist an extension model which uses type-1 and type-2 linear approximations to decrease the number of samples needed to recover the secret key in Feistel ciphers.

Open Problem III: Can we apply the DCTA attack on the Feistel cipher to SPN block cipher ?

Chapter 7 has shown that the DCTA is the most efficient attack of cache trace for Feistel block ciphers. Assume that a SPN block cipher having m identical S-boxes is considered and its hardware implementation uses only one Table Look Up (TLU) for those m S-boxes. To support the speed of encryption, the cipher has a cache. Suppose that we can observe the cache hits and cache misses during the encryption. Can we develop a DCTA attack to this cipher?

Open Problem IV: Developing an algorithm to enable 3-share TI to 8-bit S-box of *AES* block cipher.

The 3-share TI is the most efficient countermeasure of TI against the first order DPA attack. In the chapters 8 and 9, the new implementation methodology and Factorization structure are introduced to optimize and enable 3-share TI to any 4-bit S-box. Notably, those results imply that the ANF of any n -bit Sbox can be constructed by using only quadratic vectorial boolean functions. The most difficult task is that can we find a set of quadratic vectorial boolean functions such that its combination results in ANF of the given n -bit Sbox and its 3-share version is a permutation, i.e., $3n$ -bit permutation. The important application is implementing the 3-share TI of 8-bit S-box of *AES*.

BIBLIOGRAPHY

- [1] Draft Report of SA3 38. 3GPP. 2005. Technical report. Available via www.3gpp.org/ftp/tsg_sa/TSG_SA/TSGS_28/Docs/pdf/SP-050236.pdf.
- [2] ETSI. TS 135 202 V7.0.0: Universal Mobile Telecommunications System (UMTS); Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification (3GPP TS 35.202 version 7.0.0 Release 7). Technical report. Available via www.etsi.org.
- [3] ETSI. TS 55.216 V6.2.0: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the A5/3 Encryption Algorithms for GSM and ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 1: A5/3 and GEA3 Specifications (Release 6). Technical report. Available via www.etsi.org.
- [4] ETSI. TS 55.919 V6.1.0: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the A5/3 Encryption Algorithms for GSM and ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 4: Design and evaluation report (Release 6). Technical report. Available via www.etsi.org.
- [5] Specification of SMS4 Block Cipher for WLAN Products (2006)(in Chinese). Available via www.oscca.gov.cn/UpFile/200621016423197990.pdf.

-
- [6] ZUC rump session ASIACRYPT 2010. Available via www.spms.ntu.edu.sg/Asiacrypt2010/Rump%20Session-%207%20Dec%202010/wu_rump_zuc.pdf.
 - [7] ZUC Specification. Technical report. Available via www.gsmworld.com/documents/EEA3_EIA3_ZUC_v1_5.pdf.
 - [8] SKIPJACK and KEA Algorithm Specifications. Technical report, May 1998. Available via www.csrc.nist.gov/groups/STM/cavp/documents/skipjack/skipjack.pdf.
 - [9] O. Aciicmez and K.C. Koc. Trace-Driven Cache Attacks on AES (Short Paper). In P. Ning, S. Qing, and Li. N., editors, *ICICS2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 112-121. Springer-Verlag, 2006.
 - [10] E. Barkan, E. Biham, and N. Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *Journal of Cryptology*, 21(3):392-429, 2008.
 - [11] D. Bernstein. Cache-timing attacks on AES. Technical report, 2005. Available via www.cr.yp.to/antiforgery/cachetiming-20050414.pdf.
 - [12] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo. AES Power Attack Based on Induced Cache Miss and Countermeasure. In *ITCC2005*, volume 1 of *IEEE Computer Society*, pages 586-591. Springer-Verlag, 2005.
 - [13] E. Biham and A. Shamir. *Differential cryptanalysis of the data encryption standard*. Springer-Verlag, 1993.
 - [14] E. Biham, R. Anderson, and L.R. Knudsen. Serpent: A New Block Cipher Proposal. In S. Vaudenay, editor, *FSE1998*, volume 1372 of *Lecture Notes in Computer Science*, pages 222-238. Springer-Verlag, 1998.

-
- [15] E. Biham, O. Dunkelman, and N. Keller. The Rectangle Attack - Rectangling the Serpent. In B. Pfitzmann, editor, *EUROCRYPT2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 340-357. Springer-Verlag, 2001.
 - [16] E. Biham, O. Dunkelman, and N. Keller. Linear Cryptanalysis of Reduced Round Serpent. In M. Matsui, editor, *FSE2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 16-27. Springer-Verlag, 2001.
 - [17] E. Biham, O. Dunkelman, and N. Keller. A Related-Key Rectangle Attack on the Full KASUMI. In B.K. Roy, editor, *ASIACRYPT2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 443-461. Springer-Verlag, 2005.
 - [18] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In A. Menezes and S.A. Vanstone, editors, *CRYPTO1990*, volume 537 of *Lecture Notes in Computer Science*, pages 2-21. Springer-Verlag, 1990.
 - [19] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-round DES. In E.F. Brickel, editor, *CRYPTO1992*, volume 740 of *Lecture Notes in Computer Science*, pages 487-496. Springer-Verlag, 1992.
 - [20] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. Threshold Implementations of All 3×3 and 4×4 S-Boxes. In E. Prouff and P. Schaumont, editors, *CHES2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 76-91. Springer-Verlag, 2012.
 - [21] A. Biryukov, C.D. Canniere, and M. Quisquater. On Multiple Linear Approximations. In M. Franklin, editor, *CRYPTO2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 2-21. Springer-Verlag, 2004.
 - [22] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of A5/1

-
- on a PC. In B. Schneier, editor, *FSE2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 37-44. Springer-Verlag, 2000.
- [23] M. Blunden and A. Escott. Related Key Attacks on reduced round KASUMI. In M. Matsui, editor, *FSE2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 277-285. Springer-Verlag, 2001.
- [24] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *CHES2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450-466. Springer-Verlag, 2007.
- [25] J.Y. Cho, M. Hermelin, and K. Nyberg. A New Technique for Multidimensional Linear Cryptanalysis with Applications on Reduced Round Serpent. In P. Lee and J. Cheon, editors, *ICISC2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 383-398. Springer-Verlag, 2008.
- [26] B. Collard, F.X. Standaert, and J.J. Quisquater. Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent - Description of the Linear Approximations, 2007. Unpublished manuscript.
- [27] B. Collard, F.X. Standaert, and J.J. Quisquater. Improving the Time Complexity of Matsui's Linear Cryptanalysis. In K.H. Nam and G. Rhee, editors, *ICISC2007*, volume 4817 of *Lecture Notes in Computer Science*, pages 77-88. Springer-Verlag, 2007.
- [28] B. Collard, F.X. Standaert, and J.J. Quisquater. Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent. In D. Pei, M. Yung, D. Lin, and C. Wu, editors, *INSCRYPT2007*, volume 4990 of *Lecture Notes in Computer Science*, pages 51-65. Springer-Verlag, 2007.

-
- [29] B. Collard, F.X. Standaert, and J.J. Quisquater. Experiments on the Multiple Linear Cryptanalysis of Reduced Round Serpent. In K. Nyberg, editor, *FSE2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 382-397. Springer-Verlag, 2008.
- [30] T.H. Cormen, C. Stein, R. Rivest, and C.E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
- [31] Sony Corporation. The 128-bit blockcipher CLEFIA: Algorithm specification. Technical report, 2007. Available via <http://www.sony.net/Products/cryptography/clefia/download/data/clefia-design-1.0.pdf>.
- [32] W. Diffie and G. Ledin (translators). SMS4 Encryption Algorithm for Wireless Networks.
- [33] O. Dunkelman, S. Indesteege, and N. Keller. A Differential-Linear Attack on 12-Round Serpent. In D.R. Chowdhury, V. Rijmen, and Das. A., editors, *INDOCRYPT2008*, volume 5365 of *Lecture Notes in Computer Science*, pages 308-321. Springer-Verlag, 2008.
- [34] O. Dunkelman, N. Keller, and A. Shamir. A Practical-time Related-Key Attack on the KASUMI Cryptosystem used in GSM and 3G telephony. In T. Rabin, editor, *CRYPTO2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 393-410. Springer-Verlag, 2010.
- [35] D. Engels, M.J.O. Saarinen, P. Schweitzer, and E.M. Smith. The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. In A. Juels and C. Paar, editors, *RFIDSec2011*, volume 7055 of *Lecture Notes in Computer Science*, pages 19-31. Springer-Verlag, 2011.

-
- [36] X. Fan, H. Hu, G. Gong, E. M. Smith, and D. Engels. Lightweight Implementation of Hummingbird Cryptographic Algorithm on 4-Bit Microcontroller. In *ICITST 2009*, 2009.
- [37] P. Fiser and J. Hlavicka. Two-Level Boolean Minimizer BOOM-II. In *IWSBP2004*, pages 221-228, 2004.
- [38] P. Fiser and J. Hlavicka. BOOM - A Heuristic Boolean Minimizer. *Computers and Informatics*, 22(1):19-51, 2003.
- [39] J. Fournier and M. Tunstall. Cache Based Power Analysis Attacks on AES. In L.M. Batten and R.S. Naini, editors, *ACISP2006*, volume 4058 of *Lecture Notes in Computer Science*, pages 17-28. Springer-Verlag, 2006.
- [40] J.F. Gallais, I. Kizhvatov, and M. Tunstall. Improved Trace-Driven Cache-Collision Attacks against Embedded AES Implementations. In Y. Chung and M. Yung, editors, *WISA2010*, volume 6513 of *Lecture Notes in Computer Science*, pages 243-257. Springer-Verlag, 2010.
- [41] J. Guo, T. Peyrin, A. Poschmann, and M.J.B. Robshaw. The LED Block Cipher. In B. Preneel and T. Takagi, editors, *CHES2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 326-341. Springer-Verlag, 2011.
- [42] C. Harpes, G.G. Kramer, and J.L. Massey. A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-up Lemma. In L.C. Guillou and J.J. Quisquater, editors, *EUROCRYPT1995*, volume 921 of *Lecture Notes in Computer Science*, pages 24-38. Springer-Verlag, 1995.
- [43] M. Hermelin, J.Y. Cho, and K. Nyberg. Multidimensional Linear Cryptanalysis of Reduced Round Serpent. In Y. Mu, W. Susilo, and J. Seberry, editors,

-
- ACISP2008*, volume 5107 of *Lecture Notes in Computer Science*, pages 203-215. Springer-Verlag, 2008.
- [44] M. Hermelin, J.Y. Cho, and K. Nyberg. Multidimensional Extension of Matsui's Algorithm 2. In O. Dunkelman, editor, *FSE2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 209-227. Springer-Verlag, 2009.
- [45] M. Hermelin and K. Nyberg. Dependent Linear Approximations: The Algorithm of Biryukov and Others Revisited. In J. Pieprzyk, editor, *CT-RSA2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 318-333. Springer-Verlag, 2010.
- [46] H.M. Heys. A Tutorial on Linear and Differential Cryptanalysis. Technical report, 2001.
- [47] Virtual Silicon Inc. 0.18 μm VIP Standard Cell Library Tape Out Ready, Part Number: UMCL18G212T3, Process: UMC Logic 0.18 μm Generic II Technology: 0.18 μm .
- [48] N. Jacobson. *Basic Algebra*. 1 (2nd ed.). Dover. ISBN 978-0-486-47189-1.
- [49] B. Kaliski and M.J.B. Robshaw. Linear Cryptanalysis Using Multiple Approximations. In Y. Desmelt, editor, *CRYPTO1994*, volume 839 of *Lecture Notes in Computer Science*, pages 26-39. Springer-Verlag, 1994.
- [50] J. Kelsey, T.T. Kohno, and B. Schneier. Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In M. Matsui, editor, *FSE2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 75-93. Springer-Verlag, 2001.
- [51] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side Channel Cryptanalysis of Product Ciphers. *J. Comput. Secur.*, 8(2,3):141-158, 2000.

-
- [52] J. Kim, S. Hong, B. Preneel, E. Biham, O. Dunkelman, and N. Keller. Related-Key Boomerang and Rectangle Attacks.
 - [53] P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Kobitz, editor, *CRYPTO1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104-113. Springer-Verlag, 1996.
 - [54] P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M.J. Weiner, editor, *CRYPTO1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388-397. Springer-Verlag, 1999.
 - [55] U. Kühn. Cryptanalysis of Reduced-Round MISTY. In B. Pfitzmann, editor, *EUROCRYPT2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 61-75. Springer-Verlag, 2001.
 - [56] U. Kühn. Improved Cryptanalysis of MISTY1. In J. Deamen and V. Rijmen, editors, *FSE2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 61-75. Springer-Verlag, 2002.
 - [57] G. Leander, C. Paar, A. Poschmann, and K. Schramm. New Lightweight DES Variants. In A. Biryukov, editor, *FSE2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 196-210. Springer-Verlag, 2007.
 - [58] G. Leander and A. Poschmann. On the Classification of 4 Bit S-Boxes. In C. Carlet and B. Sunar, editors, *WAIFI2007*, volume 4547 of *Lecture Notes in Computer Science*, pages 159-167. Springer-Verlag, 2007.
 - [59] S. Levy. *Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age*. Viking Penguin, 2001.

-
- [60] C.H. Lim and T. Korkishko. mCrypton: a lightweight block cipher for security of low-cost RFID tags and sensors. In *WISA '05*, volume 3786 of *Lecture Notes in Computer Science*, pages 243-258. Springer-Verlag, 2006.
- [61] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, 2007.
- [62] S. Mangard, T. Popp, and B.M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *CT-RSA2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351-365. Springer-Verlag, 2005.
- [63] S. Mangard, N. Pramstaller, and E. Oswald. Successfully attacking masked AES hardware implementations. In J.R. Rao and B. Sunar, editors, *CHES2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157-171. Springer-Verlag, 2005.
- [64] M. Matsui. New Block Encryption Algorithm MISTY. In E. Biham, editor, *FSE1997*, volume 1267 of *Lecture Notes in Computer Science*, pages 54-68. Springer-Verlag, 1997.
- [65] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseeth, editor, *EUROCRYPT1993*, volume 765 of *Lecture Notes in Computer Science*, pages 386-397. Springer-Verlag, 1993.
- [66] M. Matsui. The First Experimental Cryptanalysis of the Data Encryption Standard. In Y. Desmedt, editor, *CRYPTO1994*, volume 839 of *Lecture Notes in Computer Science*, pages 1-11. Springer-Verlag, 1994.
- [67] M. Matsui and Yamagishi. A. A New Method for Known Plaintext Attack

-
- of FEAL cipher. In R.A. Rueppel, editor, *EUROCRYPT1992*, volume 765 of *Lecture Notes in Computer Science*, pages 81-91. Springer-Verlag, 1992.
- [68] A. Menezes, P.V. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.
- [69] A. Moradi. Statistical Tools Flavor Side-Channel Collision Attacks. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 428-445. Springer-Verlag, 2012.
- [70] A. Moradi, A. Poschmann, C. Paar, and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In K.G. Paterson, editor, *EUROCRYPT2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69-88. Springer-Verlag, 2011.
- [71] U.S Department of Commerce National Bureau of Standards. Data Encryption Standard. Technical report, 1977. Available via <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>.
- [72] P.H. Nguyen, L. Wei, H. Wang, and S. Ling. On Multidimensional Linear Cryptanalysis. In R. Steinfeld and P. Hawkes, editors, *ACISP2010*, volume 6168 of *Lecture Notes in Computer Science*, pages 37-52. Springer-Verlag, 2010.
- [73] S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In P. Ning, S. Quing, and N. Li, editors, *ICICS2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 529-545. Springer-Verlag, 2006.
- [74] S. Nikova, V. Rijmen, and M. Schlffer. Secure Hardware Implementation

-
- of Nonlinear Functions in the Presence of Glitches. *Journal of Cryptology*, 24(2):292-321, 2011.
- [75] S. Nikova, V. Rijmen, and M. Schlffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In P. Lee and J. Cheon, editors, *ICISC2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218-234. Springer-Verlag, 2008.
- [76] K. Nyberg. Linear Approximation of Block Ciphers. In A.D. Santis, editor, *EUROCRYPT1994*, volume 950 of *Lecture Notes in Computer Science*, pages 439-444. Springer-Verlag, 1994.
- [77] D. Page. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel.
- [78] R. Poddar, A. Datta, and C. Rebeiro. A Cache Trace Attack on CAMELLIA. In M. Joyce, D. Mukhopadhyay, and M. Tunstall, editors, *InfoSecHiComNet2011*, volume 7011 of *Lecture Notes in Computer Science*, pages 144-156. Springer-Verlag, 2011.
- [79] T. Popp and S. Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In J.R. Rao and B. Sunar, editors, *CHES2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 172-186. Springer-Verlag, 2005.
- [80] A. Poschmann, A. Moradi, K. Khoo, C. Lim, C. Wee, H. Wang, and S. Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. *Journal of Cryptology*, 24(2):322-345, 2011.
- [81] C. Rebeiro and D. Mukhopadhyay. Cryptanalysis of CLEFIA Using Differential Methods with Cache Trace Patterns. In A. Kiayias, editor, *CT-RSA2011*, vol-

-
- ume 6558 of *Lecture Notes in Computer Science*, pages 89-103. Springer-Verlag, 2011.
- [82] C. Rebeiro, R. Poddar, A. Datta, and D. Mukhopadhyay. An Enhanced Differential Cache Attack on CLEFIA for Large Cache Lines. In D. Bernstein and S. Chatterjee, editors, *INDOCRYPT2011*, volume 7107 of *Lecture Notes in Computer Science*, pages 58-75. Springer-Verlag, 2011.
- [83] R.L. Rivest. The RC5 Encryption Algorithm. In B. Preneel, editor, *FSE1994*, volume 1008 of *Lecture Notes in Computer Science*, pages 86-96. Springer-Verlag, 1994.
- [84] M.J.O. Saarinen. Cryptographic Analysis of All 4×4 S-boxes. In A. Miri and S. Vaudenay, editors, *SAC2011*, volume 7118 of *Lecture Notes in Computer Science*, pages 118-133. Springer-Verlag, 2011.
- [85] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. The Twofish Encryption Algorithm. Technical report, 1998.
- [86] C. Shannon. Communication Theory of Secrecy System. *Bell System Technical Journal*, 28:656-715, 1949.
- [87] A. Sorkin. Lucifer, a cryptographic algorithm. *Cryptologia*, 8(1):22-41, 1984.
- [88] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi. TWINE: A Lightweight Block Cipher for Multiple Platforms. In H. Knudsen, L.R. and Wu, editor, *SAC2012*, volume 7707 of *Lecture Notes in Computer Science*, pages 339-354. Springer-Verlag, 2013.
- [89] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi. Cryptanalysis of DES Implemented on Computers with Cache. In C.D. Walter, C.K. Koc,

-
- and C. Paar, editors, *CHES2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 62-76. Springer-Verlag, 2003.
- [90] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi. Cryptanalysis of Block Ciphers Implemented on Computers with Cache. In *ISITA2002*, 2002.
- [91] H.V. Tung and P. Rogaway. On generalized Feistel networks. In T. Rabin, editor, *CRYPTO2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 613-630. Springer-Verlag, 2010.
- [92] J. Waddle and D. Wagner. Towards Efficient Second-Order Power Analysis. In M. Joye and J.J. Quisquater, editors, *CHES2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 1-15. Springer-Verlag, 2004.
- [93] D. Wagner. The Boomerang Attack. In L.R. Knudsen, editor, *FSE1999*, volume 1636 of *Lecture Notes in Computer Science*, pages 156-170. Springer-Verlag, 1999.
- [94] X. Wang, K.H.L. Chi, C.F. Chong, K.P. Chow, W.W. Tsang, and H.W. Chen. The Differential Cryptanalysis of an AES Finalist-Serpent, 2000. Available via www.cs.hku.hk/publications/techreps/document/TR-2000-04.ps.
- [95] K.R. Yarlagadda and J. Hershey. *Hadamard Matrix Analysis and Synthesis: with Applications to Communications and Signal/Image Processing*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.
- [96] I.A. Zabolotin, G.P. Glazkov, and V.B. Isaeva. Cryptographic Protection for Information Processing Systems, Government Standard of the USSR, GOST 28147-89, Government Committee of the USSR for Standards. Technical report, 1989.

-
- [97] L. Zhang, W. Zhang, and W. Wu. Cryptanalysis of Reduced-Round SMS4 Block Cipher. In Y. Mu, W. Susilo, and J. Seberry, editors, *ACISP2008*, volume 5107 of *Lecture Notes in Computer Science*, pages 216-229. Springer-Verlag, 2008.
- [98] W. Zhang, W. Wu, D. Feng, and B. Su. Some New Observations on the SMS4 Block Cipher in the Chinese WAPI Standard. In B. Feng, L. Hui, and G. Wang, editors, *ISPEC2009*, volume 5451 of *Lecture Notes in Computer Science*, pages 324-335. Springer-Verlag, 2009.

APPENDIX

A. CRYPTANALYSIS OF ZUC STREAM CIPHER

k_{15}	k_0	$\# \{k_{13}, k_{10}, k_4\}$	k_{15}	k_0	$\# \{k_{13}, k_{10}, k_4\}$	k_{15}	k_0	$\# \{k_{13}, k_{10}, k_4\}$	k_{15}	k_0	$\# \{k_{13}, k_{10}, k_4\}$
0x94	0x21	262144	0x94	0x23	262144	0x94	0x25	262144	0x94	0x27	262144
0x94	0x29	262144	0x94	0x2b	262144	0x94	0x2d	262143	0x94	0x2f	262144
0x94	0x31	262144	0x94	0x33	262144	0x94	0x35	262144	0x94	0x37	262144
0x94	0x39	262144	0x94	0x3b	262144	0x94	0x3d	262144	0x94	0x3f	262144
0x94	0x45	262144	0x94	0x47	262144	0x94	0x49	262144	0x94	0x4b	262144
0x94	0x4d	262144	0x94	0x4f	262144	0x94	0x61	262144	0x94	0x63	262144
0x94	0x65	262144	0x94	0x67	262144	0x94	0x69	262144	0x94	0x6b	262144
0x94	0x6d	262144	0x94	0x6f	262144	0x94	0x71	262144	0x94	0x73	262144
0x94	0x75	262144	0x94	0x77	262144	0x94	0x79	262144	0x94	0x7b	262144
0x94	0x7d	262144	0x94	0x7f	262144	0x94	0x81	262144	0x94	0x83	262144
0x94	0x85	262144	0x94	0x87	262144	0x94	0x89	262144	0x94	0x8b	262144
0x94	0x8d	157696	0x94	0x91	262144	0x94	0x93	262144	0x94	0x95	262144
0x94	0x97	262144	0x94	0x99	262144	0x94	0x9b	262144	0x94	0x9d	262144
0x94	0x9f	262144	0x94	0xc1	262144	0x94	0xc3	262144	0x94	0xc5	262144
0x94	0xc7	262144	0x94	0xc9	262144	0x94	0xcb	262144	0x94	0xcd	262144
0x94	0xcf	262144	0x94	0xd1	262144	0x94	0xd3	262144	0x94	0xd5	262144
0x94	0xd7	262144	0x94	0xd9	262144	0x94	0xdb	262144	0x94	0xdd	262144
0x94	0xdf	262144	0x94	0xe1	262144	0x94	0xe3	262144	0x94	0xe5	262144
0x94	0xe7	262144	0x94	0xe9	262144	0x94	0xeb	262144	0x94	0xed	262144
0x94	0xef	262144	0x94	0xf1	262144	0x94	0xf3	262144	0x94	0xf5	262144
0x94	0xf7	262144	0x94	0xf9	262144	0x94	0xfb	262144	0x94	0xfd	262144
0x94	0xff	262144	0xeb	0x01	262144	0xeb	0x03	262144	0xeb	0x05	262144
0xeb	0x07	262144	0xeb	0x09	262144	0xeb	0x0b	262144	0xeb	0x0d	262144
0xeb	0x0f	262144	0xeb	0x11	262144	0xeb	0x13	262144	0xeb	0x15	262144
0xeb	0x17	262144	0xeb	0x19	262144	0xeb	0x1b	262144	0xeb	0x1d	262144
0xeb	0x1f	262144	0xeb	0x21	262144	0xeb	0x23	262144	0xeb	0x25	262144
0xeb	0x27	262144	0xeb	0x29	262144	0xeb	0x2b	262144	0xeb	0x2d	262144
0xeb	0x2f	262144	0xeb	0x31	262144	0xeb	0x33	262144	0xeb	0x35	262144
0xeb	0x37	262144	0xeb	0x45	262144	0xeb	0x47	262144	0xeb	0x49	262144
0xeb	0x4b	262144	0xeb	0x4d	262144	0xeb	0x61	262144			

Tab. A.1: The list of weak keys with IV_0 difference of ZUC.

$$SET_D = \{\# \{iv_{15}, iv_4\} : \text{cover all 453 filtered values}\} = \{$$

$(0x1a, 0x00), (0x1f, 0x00), (0x38, 0x00), (0x45, 0x00), (0x7c, 0x00),$
 $(0x8f, 0x00), (0x94, 0x00), (0xd7, 0x00), (0xd9, 0x00), (0x05, 0x01),$
 $(0x2f, 0x01), (0x6b, 0x01), (0x77, 0x01), (0x87, 0x01), (0x90, 0x01),$
 $(0x94, 0x01), (0x95, 0x01), (0x96, 0x01), (0x97, 0x01), (0x98, 0x01),$
 $(0x99, 0x01), (0x9a, 0x01), (0x9b, 0x01), (0x9c, 0x01), (0x9d, 0x01),$

(0x9e,0x01), (0x9f,0x01), (0xa0,0x01), (0xa1,0x01), (0xa2,0x01),
 (0xa3,0x01), (0xa4,0x01), (0xa5,0x01), (0xa6,0x01), (0xa7,0x01),
 (0xa8,0x01), (0xa9,0x01), (0xaa,0x01), (0xab,0x01), (0xac,0x01),
 (0xad,0x01), (0xae,0x01), (0xaf,0x01), (0xb0,0x01), (0xb1,0x01),
 (0xb2,0x01), (0xb3,0x01), (0xb4,0x01), (0xb5,0x01), (0xb6,0x01),
 (0xb7,0x01), (0xb8,0x01), (0xb9,0x01), (0xba,0x01), (0xbb,0x01),
 (0xbc,0x01), (0xbd,0x01), (0xbe,0x01), (0xbf,0x01), (0xc0,0x01),
 (0xc1,0x01), (0xc2,0x01), (0xc3,0x01), (0xc4,0x01), (0xc5,0x01),
 (0xc6,0x01), (0xdf,0x01), (0xf9,0x01), (0x06,0x02), (0x37,0x02),
 (0x4c,0x02), (0x52,0x02), (0x80,0x02), (0x9b,0x02), (0xde,0x02),
 (0xf1,0x02), (0x07,0x03), (0x84,0x03), (0x8f,0x03), (0xa5,0x03),
 (0xca,0x03), (0xd6,0x03), (0xe5,0x03), (0xe6,0x03), (0xea,0x03),
 (0x19,0x04), (0x39,0x04), (0x52,0x04), (0xc3,0x04), (0xdc,0x04),
 (0xec,0x04), (0x0d,0x05), (0x3b,0x05), (0xcd,0x05), (0xda,0x05),
 (0xf5,0x05), (0x13,0x06), (0x38,0x06), (0x46,0x06), (0x86,0x06),
 (0x9c,0x06), (0xa5,0x06), (0xda,0x06), (0xe0,0x06), (0xfc,0x06),
 (0x21,0x07), (0x37,0x07), (0x3b,0x07), (0x4f,0x07), (0x95,0x07),
 (0x96,0x07), (0x97,0x07), (0x98,0x07), (0x99,0x07), (0x9a,0x07),
 (0x9b,0x07), (0x9c,0x07), (0x9d,0x07), (0x9e,0x07), (0x9f,0x07),
 (0xa0,0x07), (0xa1,0x07), (0xa2,0x07), (0xa3,0x07), (0xa4,0x07),
 (0xa5,0x07), (0xa6,0x07), (0xa7,0x07), (0xa8,0x07), (0xa9,0x07),
 (0xaa,0x07), (0xab,0x07), (0xac,0x07), (0xad,0x07), (0xae,0x07),
 (0xaf,0x07), (0xb0,0x07), (0xb1,0x07), (0xb2,0x07), (0xb3,0x07),
 (0xb4,0x07), (0xb5,0x07), (0xb6,0x07), (0xb7,0x07), (0xb8,0x07),
 (0xb9,0x07), (0xba,0x07), (0xbb,0x07), (0xbc,0x07), (0xbd,0x07),
 (0xbe,0x07), (0xbf,0x07), (0xc0,0x07), (0xc1,0x07), (0xc2,0x07),

(0xc3,0x07), (0xc4,0x07), (0xc5,0x07), (0xc6,0x07), (0xe2,0x07),
 (0xe4,0x07), (0x0d,0x08), (0x11,0x08), (0x4f,0x08), (0x71,0x08),
 (0x96,0x08), (0x9a,0x08), (0xbf,0x08), (0xf2,0x08), (0xf9,0x08),
 (0xfb,0x08), (0x4a,0x09), (0x56,0x09), (0x68,0x09), (0x6a,0x09),
 (0x82,0x09), (0x95,0x09), (0x96,0x09), (0x97,0x09), (0x98,0x09),
 (0x99,0x09), (0x9a,0x09), (0x9b,0x09), (0x9c,0x09), (0x9d,0x09),
 (0x9e,0x09), (0x9f,0x09), (0xa0,0x09), (0xa1,0x09), (0xa2,0x09),
 (0xa3,0x09), (0xa4,0x09), (0xa5,0x09), (0xa6,0x09), (0xa7,0x09),
 (0xa8,0x09), (0xa9,0x09), (0xaa,0x09), (0xab,0x09), (0xac,0x09),
 (0xad,0x09), (0xae,0x09), (0xaf,0x09), (0xb0,0x09), (0xb1,0x09),
 (0xb2,0x09), (0xb3,0x09), (0xb4,0x09), (0xb5,0x09), (0xb6,0x09),
 (0xb7,0x09), (0xb8,0x09), (0xb9,0x09), (0xba,0x09), (0xbb,0x09),
 (0xbc,0x09), (0xbd,0x09), (0xbe,0x09), (0xbf,0x09), (0xc0,0x09),
 (0xc1,0x09), (0xc2,0x09), (0xc3,0x09), (0xc4,0x09), (0xc5,0x09),
 (0xc6,0x09), (0xcc,0x09), (0xf0,0x09), (0x0e,0x0a), (0x89,0x0a),
 (0x92,0x0a), (0xac,0x0a), (0xb6,0x0a), (0xb9,0x0a), (0xe3,0x0a),
 (0xe7,0x0a), (0xec,0x0a), (0x35,0x0b), (0x3b,0x0b), (0xa9,0x0b),
 (0xbb,0x0b), (0xbc,0x0b), (0x04,0x0c), (0x05,0x0c), (0x0c,0x0c),
 (0x2d,0x0c), (0x2f,0x0c), (0x90,0x0c), (0x14,0x0d), (0x3a,0x0d),
 (0x6f,0x0d), (0x7d,0x0d), (0x86,0x0d), (0xa0,0x0d), (0xe7,0x0d),
 (0xfc,0x0d), (0x2c,0x0e), (0x42,0x0e), (0x44,0x0e), (0x5b,0x0e),
 (0x6f,0x0e), (0x7f,0x0e), (0x89,0x0e), (0xac,0x0e), (0xb5,0x0e),
 (0xb6,0x0e), (0xcf,0x0e), (0xea,0x0e), (0xf4,0x0e), (0x2f,0x0f),
 (0x4b,0x0f), (0x52,0x0f), (0x63,0x0f), (0x87,0x0f), (0x97,0x0f),
 (0xa1,0x0f), (0xa3,0x0f), (0xb6,0x0f), (0x3b,0x10), (0x40,0x11),
 (0x46,0x11), (0x5f,0x11), (0x8d,0x11), (0x94,0x11), (0x37,0x12),

(0x93,0x12), (0xa6,0x12), (0xbe,0x12), (0xe7,0x12), (0x25,0x13),
 (0x5c,0x13), (0x6e,0x13), (0xa2,0x13), (0xb3,0x13), (0x2e,0x14),
 (0x38,0x14), (0x5d,0x14), (0x75,0x14), (0x8d,0x14), (0xfe,0x14),
 (0x1b,0x15), (0x41,0x15), (0x4a,0x15), (0x5b,0x15), (0x60,0x15),
 (0x78,0x15), (0xbf,0x15), (0xce,0x15), (0x1a,0x16), (0x1d,0x16),
 (0x2b,0x16), (0x2c,0x16), (0xc1,0x16), (0xd0,0x16), (0xfa,0x16),
 (0xff,0x16), (0x35,0x17), (0x3b,0x17), (0x68,0x17), (0x92,0x17),
 (0x98,0x17), (0x9a,0x17), (0xa4,0x17), (0xc4,0x17), (0x09,0x18),
 (0xb9,0x18), (0xce,0x18), (0xff,0x18), (0x1d,0x19), (0x46,0x19),
 (0x61,0x19), (0xbe,0x19), (0xfc,0x19), (0x00,0x1a), (0x08,0x1a),
 (0x58,0x1a), (0xb0,0x1a), (0xc3,0x1a), (0xf6,0x1a), (0x6e,0x1b),
 (0x9f,0x1b), (0xb0,0x1b), (0x0f,0x1c), (0x8d,0x1c), (0x9b,0x1c),
 (0xeb,0x1c), (0x20,0x1d), (0x45,0x1d), (0x96,0x1d), (0x97,0x1d),
 (0x98,0x1d), (0x99,0x1d), (0x9a,0x1d), (0x9b,0x1d), (0x9c,0x1d),
 (0x9d,0x1d), (0x9e,0x1d), (0x9f,0x1d), (0xa0,0x1d), (0xa1,0x1d),
 (0xa2,0x1d), (0xa3,0x1d), (0xa4,0x1d), (0xa5,0x1d), (0xa6,0x1d),
 (0xa7,0x1d), (0xa8,0x1d), (0xa9,0x1d), (0xaa,0x1d), (0xab,0x1d),
 (0xac,0x1d), (0xad,0x1d), (0xae,0x1d), (0xaf,0x1d), (0xb0,0x1d),
 (0xb1,0x1d), (0xb2,0x1d), (0xb3,0x1d), (0xb4,0x1d), (0xb5,0x1d),
 (0xb6,0x1d), (0xb7,0x1d), (0xb8,0x1d), (0xb9,0x1d), (0xba,0x1d),
 (0xbb,0x1d), (0xbc,0x1d), (0xbd,0x1d), (0xbe,0x1d), (0xbf,0x1d),
 (0xc0,0x1d), (0xc1,0x1d), (0xc2,0x1d), (0xc3,0x1d), (0xc4,0x1d),
 (0xc5,0x1d), (0xc6,0x1d), (0x44,0x1e), (0xd0,0x1e), (0x32,0x1f),
 (0x6f,0x1f), (0x96,0x1f), (0x97,0x1f), (0x98,0x1f), (0x99,0x1f),
 (0x9a,0x1f), (0x9b,0x1f), (0x9c,0x1f), (0x9d,0x1f), (0x9e,0x1f),
 (0x9f,0x1f), (0xa0,0x1f), (0xa1,0x1f), (0xa2,0x1f), (0xa3,0x1f),

$(0xa4, 0x1f), (0xa5, 0x1f), (0xa6, 0x1f), (0xa7, 0x1f), (0xa8, 0x1f),$
 $(0xa9, 0x1f), (0xaa, 0x1f), (0xab, 0x1f), (0xac, 0x1f), (0xad, 0x1f),$
 $(0xae, 0x1f), (0xaf, 0x1f), (0xb0, 0x1f), (0xb1, 0x1f), (0xb2, 0x1f),$
 $(0xb3, 0x1f), (0xb4, 0x1f), (0xb5, 0x1f), (0xb6, 0x1f), (0xb7, 0x1f),$
 $(0xb8, 0x1f), (0xb9, 0x1f), (0xba, 0x1f), (0xbb, 0x1f), (0xbc, 0x1f),$
 $(0xbd, 0x1f), (0xbe, 0x1f), (0xbf, 0x1f), (0xc0, 0x1f), (0xc1, 0x1f),$
 $(0xc2, 0x1f), (0xc3, 0x1f), (0xc4, 0x1f), (0xc5, 0x1f), (0xc6, 0x1f),$
 $(0xc7, 0x1f), (0xe8, 0x1f), (0xff, 0x1f).$

}

B. ALGORITHM 1 OF MULTIDIMENSIONAL LINEAR CRYPTANALYSIS

B.1 Proof For Lemma 4.3.1

Proof. Let $g = (g_1, \dots, g_m)$ and given $\rho(a)$, for all $a \in V_m$, from (2), (3)

$$2^m(\sum p_\eta^2) = \sum \rho^2(a).$$

We have 2 facts: $p^0 = p$ and p^w is a permutation of p . From (1), we replace p^0 and p^w by p, q respectively in (5). Define $u = \max\{q_\eta, \eta \in V_m\} = \max\{p_\eta, \eta \in V_m\}$. Then

$$C(p||q) = \sum_{\eta \in V_m} \frac{(p_\eta - q_\eta)^2}{q_\eta} \geq \frac{1}{u} \sum_{\eta \in V_m} (p_\eta^2 + q_\eta^2 - 2p_\eta q_\eta).$$

We have

$$\frac{q_\eta}{u} \leq 1 \Rightarrow -\frac{q_\eta}{u} \geq -1.$$

Since q is a permutation of p :

$$\sum_{\eta \in V_m} p_\eta = 1 \text{ and } \sum_{\eta \in V_m} p_\eta^2 = \sum_{\eta \in V_m} q_\eta^2.$$

Hence

$$\begin{aligned}
\frac{1}{u} \sum_{\eta \in V_m} (p_\eta^2 + q_\eta^2 - 2p_\eta q_\eta) &= \frac{2}{u} \left(\sum_{\eta \in V_m} p_\eta^2 \right) - 2 \sum_{\eta \in V_m} \frac{q_\eta}{u} p_\eta \\
&\geq \frac{2 \cdot 2^{-m} \sum_{a \in V_m} \rho^2(a)}{u} - 2 \left(\sum_{\eta \in V_m} p_\eta \right) \\
&= \frac{2 \cdot 2^{-m} \sum_{a \in V_m} \rho^2(a)}{u} - 2.
\end{aligned}$$

From (3),

$$u = \max\{p_\eta\} \leq 2^{-m} \left(\sum_{a \in V_m} |\rho(a)| \right),$$

so that

$$C(p||q) \geq \frac{2 \sum_{a \in V_m} \rho^2(a)}{\sum_{a \in V_m} |\rho(a)|} - 2.$$

In practice, since $\rho(0) = 1$ and $\rho(a) \ll 1$, $\forall a \neq 0$, we have $\sum_{a \in V_m} |\rho(a)| \approx 1$.

Hence

$$C(p||q) \geq 2 \sum_{\forall a \in V_m \setminus \{0\}} \rho^2(a).$$

■

C. THE ALGORITHM 2 OF MULTIDIMENSIONAL LINEAR CRYPTANALYSIS

C.1 Proof and Description of Lemma 5.3.1

Proof. The Lemma 1 and the corollary 1 show that the probability distribution p of g can be calculated from correlations $\rho(a)$ of combined approximations $a \cdot g$ for $a \in V_m$. The $a \cdot g$ combined approximations are defined as follows:

$$a \cdot g := \begin{cases} 0 & \text{if } a \text{ is a zero vector} \\ a \cdot (u_1 P, \dots, u_m P) \oplus f(K_n \oplus C_l) & a \in V_m, a \neq 0. \end{cases}$$

Let $\mathbf{r}^{2^m \times 2^l}[a, K]$ be the matrix whose elements $\mathbf{r}[a, K]$ are the correlations of linear approximations $a \cdot g$ when $K_n = K$. We have 2 following cases:

1. a is zero vector. Since $a \cdot g$ is equal to 0, $\mathbf{r}[0, K] = 1$, ($\forall K \in V_l$).
2. a is not zero vector. *method 1* below will compute $\mathbf{r}[a, K]$ ($K \in V_l$).

Since our aim is to reduce as much as possible the number of times of working with N samples for computing $\mathbf{r}[a, K]$, we will need 2 assistant tables $T^{2^m \times 2^l}$ and $E^{2^m \times 2^l}$.

In practice, the empirical $\mathbf{r}[a, K]$ is computed as follows [27]:

$$\mathbf{r}[a, K] = \frac{\#\{(P_t, C_t) : a \cdot g(P_t, C_t, K) = 0\} - \#\{(P_t, C_t) : a \cdot g(P_t, C_t, K) = 1\}}{N},$$

$$t = 1, \dots, N.$$

We go through all $a \in V_m, (P_t, C_t)$ for $t = 1, \dots, N$ to compute the value of table T . The elements of this table are computed as follows:

$$a(u_1 P_t, \dots, u_m P_t) = \begin{cases} 0 \rightarrow T(a, (C_l)_t) := T(a, (C_l)_t) + 1 \\ 1 \rightarrow T(a, (C_l)_t) := T(a, (C_l)_t) - 1, \end{cases}$$

where $(C_l)_t$ is the l bits of ciphertext C_t involved in function $f(\cdot)$, $\forall a \in V_m$ and (P_t, C_t) ($t = 1, \dots, N$).

Let $T[a, v] = T[a, v]/N$ ($a \in V_m, v \in V_l$), then

$$\mathbf{r}[a, K] = \sum_{v=0}^{2^l-1} (-1)^{f(K \oplus v)} T[a, v].$$

Let $\mathbf{S}^{2^l \times 2^l}$ be a matrix with $\mathbf{S}[i, j] = (-1)^{f(i \oplus j)}$ ($\forall i, j \in V_l$). Hence,

$$\mathbf{r}[a, \cdot] = \mathbf{S}T[a, \cdot]. \quad (\text{C.1})$$

Based on the proposition in [27], matrix $\mathbf{S}$ is a *circulant* matrix. Therefore, vector $\mathbf{r}[a, \cdot]$ is computed with complexity $3l2^l$ and we only need 2^l memory units to store the first column $((-1)^{f(0)}, \dots, (-1)^{f(2^l-1)})^T$ of matrix $\mathbf{S}$ to calculate $\mathbf{r}[a, \cdot]$.

The time complexity is very high if we directly compute T from N samples, i.e., $\mathcal{O}(2^m N)$. We show a way to compute T more efficiently by assistant table E . We

construct $\mathbf{E}^{2^m \times 2^l}$ as follows:

$$E[h_1, h_2] = \#\{(P_t, C_t), t = 1, \dots, N : (u_1 \cdot P_t, \dots, u_m \cdot P_t) = h_1 \text{ and } (C_t)_t = h_2\},$$

$$h_1 \in V_m, h_2 \in V_l.$$

The time complexity for constructing $\mathbf{E}$ is mN and the memory complexity is 2^{m+l} . If we do m computing $u_1 \cdot P_t, \dots, u_m \cdot P_t$ for each sample (P_t, C_t) at the same time, then the time complexity for constructing E is N .

For $\forall v \in V_l, \forall a \in V_m$, We have

$$T[a, v] = \sum_{h_1=0}^{2^m-1} (-1)^{ah_1} E[h_1, v]. \quad (\text{C.2})$$

Let $\mathbf{H}^{2^m \times 2^m}$ be a matrix with $H[i, j] = (-1)^{ij}$ ($\forall i, j \in V_m$). Then,

$$T[\cdot, v] = \mathbf{H}E[\cdot, v]. \quad (\text{C.3})$$

Since matrix $\mathbf{H}$ is a *Hadamard* matrix, $T[\cdot, v]$ can be computed using *FWHT* algorithm with time complexity $m2^m$ and memory complexity 2^m . ■

In summary, the steps taken in *method 1* are as follows:

1. Compute $\mathbf{E}^{2^m \times 2^l}$ with N time complexity and 2^{m+l} memory complexity.
2. For $\forall v \in V_l$, compute $T[\cdot, v] = (T[0, v], \dots, T[2^m - 1, v])$ using *FWHT* algorithm in (C.3). Time complexity is $2^l m 2^m (= m 2^{l+m})$ and memory complexity is 2^{m+l} .
3. For $\forall a \in V_m, a \neq 0$, compute $\mathbf{r}[a, \cdot]$ by the algorithm of *circulant* matrix [27] in (C.1). Time complexity is $2^m 3l 2^l (= 3l 2^{l+m})$ and memory complexity is 2^l .

4. If a is a zero vector, then $\mathbf{r}[a, \cdot]$ is vector 1.
5. We need 2^{m+l} memory units to store all $\mathbf{r}[a, K]$ ($\forall K \in V_l, \forall a \in V_m$).
6. If we need to compute $\mathbf{p}[K, \cdot]$ ($\forall K \in V_l$), then time complexity is $3m2^m2^l (= 3m2^{l+m})$ and 2^{m+l} memory units needed for all $\mathbf{p}$.

The total time complexity is $\mathcal{O}(N + 2^{l+m}(4m + 3l))$ and memory complexity is $\mathcal{O}(2^{m+l})$.

C.2 Proof and Description of Lemma 5.3.2

Proof. The combined approximations $a \cdot g$ are defined as follows:

$$a \cdot g = \begin{cases} a \cdot f_1(K_1 \oplus P_{l_1}) \oplus f_2(K_n \oplus C_{l_2}) & \text{if } a \text{ is not a zero vector} \\ 0 & \text{if } a \text{ is a zero vector} \end{cases}$$

, where $f_1(K_1 \oplus P_{l_1}) = (f_1^1(K_1 \oplus P_{l_1}), \dots, f_1^m(K_1 \oplus P_{l_1}))$, $\forall a \in V_m$.

Let $\mathbf{r}^{2^m \times 2^{l_1} \times 2^{l_2}}$ be the matrix containing all the correlations of $a \cdot g$ with $a \in V_m, K_1 \in V_{l_1}, K_n \in V_{l_2}$. We have the following 2 cases:

1. If a is vector 0, then $\mathbf{r}[a, k_1, k_2] = 1$ ($\forall k_1 \in V_{l_1}, \forall k_2 \in V_{l_2}$).
2. If a is not vector 0, then we can compute $\mathbf{r}[a, k_1, k_2]$ ($\forall k_1 \in V_{l_1}, \forall k_2 \in V_{l_2}$) by *method 2* below.

We need 2 assistant tables $T^{2^m \times 2^{l_1} \times 2^{l_2}}$ and $E^{2^{l_1} \times 2^{l_2}}$.

Let

$$\begin{aligned}\mathbf{r}[a, k_1, \cdot] &= (\mathbf{r}[a, k_1, 0], \dots, \mathbf{r}[a, k_1, 2^{l_2} - 1]), \\ T[a, k_1, \cdot] &= (T[a, k_1, 0], \dots, T[a, k_1, 2^{l_2} - 1]).\end{aligned}$$

For $\forall a \in V_m$, (P_t, C_t) ($t = 1, \dots, N$), $\forall k_1 \in V_{l_1}$, the elements of table T are calculated as follows:

$$af_1(k_1 \oplus (P_{l_1})_t) = \begin{cases} 0 \rightarrow T[a, k_1, (C_{l_2})_t] := T[a, k_1, (C_{l_2})_t] + 1 \\ 1 \rightarrow T[a, k_1, (C_{l_2})_t] := T[a, k_1, (C_{l_2})_t] - 1, \end{cases}$$

where $(P_{l_1})_t$ and $(C_{l_2})_t$ are l_1 bits of P_t and l_2 bits of C_t involved in the attack, respectively.

Let $T[a, k_1, k_2] = T[a, k_1, k_2]/N$.

With the same argument in *method 1*, we have

$$\mathbf{r}[a, k_1, k_2] = \sum_{v=0}^{2^{l_2}-1} (-1)^{f_2(k_2 \oplus v)} T[a, k_1, v].$$

Let $\mathbf{S}_2^{2^{l_2} \times 2^{l_2}}$ be a matrix with $\mathbf{S}_2[i, j] = (-1)^{f_2(i \oplus j)}$ ($\forall i, j \in V_{l_2}$). Hence,

$$\mathbf{r}[a, k_1, \cdot] = \mathbf{S}_2 T[a, k_1, \cdot]. \quad (\text{C.4})$$

According to [27], matrix $\mathbf{S}_2$ is a *circulant* matrix. Hence, vector $\mathbf{r}[a, k_1, \cdot]$ is computed with time complexity $3l_2 2^{l_2}$ and 2^{l_2} memory units.

In order to efficiently compute $T[a, k_1, v]$, we construct the table $\mathbf{E}^{2^{l_1} \times 2^{l_2}}$ as follows:

$$E[h_1, h_2] = \sharp\{(P_t, C_t), t = 1, \dots, N : (P_{l_1})_t = h_1, (C_{l_2})_t = h_2\}, \forall h_1 \in V_{l_1}, \forall h_2 \in V_{l_2}.$$

The time complexity to construct $\mathbf{E}$ is N and the memory complexity is $2^{l_1+l_2}$. Then,

$$T[a, k_1, v] = \sum_{u=0}^{2^{l_1}-1} (-1)^{af_1(k_1 \oplus u)} E[u, v]. \quad (\text{C.5})$$

Let $\mathbf{S}_a^{2^{l_1} \times 2^{l_1}}$ be a matrix with $\mathbf{S}_a[i, j] = (-1)^{af_1(i \oplus j)}$ ($\forall i, j \in V_{l_1}$). Then,

$$T[a, \cdot, k_2] = \mathbf{S}_a E[\cdot, v], \quad (\text{C.6})$$

where $E[\cdot, v]$ is the column of matrix $\mathbf{E}$.

Based on [27], matrix S_a is a *circulant* matrix. Hence, the vector $T[a, \cdot, k_2]$ is computed in $3l_1 2^{l_1}$ computations and the memory needed is 2^{l_1} . ■

In summary, the steps taken in *method 2* are as follows:

1. Construct table $\mathbf{E}$. Time complexity is N and the memory needed for $\mathbf{E}$ is $2^{l_1+l_2}$.
2. For $\forall a \in V_m, \forall v \in V_{l_2}$, we compute all $T[a, \cdot, k_2]$ by (C.6). Time complexity is $2^m 2^{l_2} 3l_1 2^{l_1} (= 3l_1 2^{l_1+l_2+m})$. We need $2^{m+l_1+l_2}$ memory units for all $T[a, k_1, v]$ ($\forall a \in V_m, \forall k_1 \in V_{l_1}, \forall v \in V_{l_2}$).
3. For $\forall a \in V_m, a \neq 0, \forall k_1 \in V_{l_1}$ compute all $\mathbf{r}[a, k_1, \cdot]$ by (C.4). The time complexity is $2^m 2^{l_1} 3l_2 2^{l_2} (= 3l_2 2^{l_1+l_2+m})$. If a is a vector zero, the $\mathbf{r}[a, k_1, \cdot]$ is the vector 1 ($\forall k_1 \in V_{l_1}$). We need $2^{l_1+l_2+m}$ memory for $\mathbf{r}$.
4. If we need to compute $\mathbf{p}$, then time complexity is $3m 2^m 2^{l_1+l_2} (= 3m 2^{l_1+l_2+m})$ and memory complexity is $2^{m+l_1+l_2}$.

Consequently, the total time complexity is $\mathcal{O}(N + 3(l_1 + l_2 + m) 2^{l_1+l_2+m})$ and the memory complexity is $\mathcal{O}(2^{m+l_1+l_2})$.